NPL Report CMSC 18/03

Report to the National Measurement System Policy Unit, Department of Trade and Industry

From the Software Support for Metrology Programme

Testing Algorithms in Standards and METROS

By
R M Barker, M G Cox, P M Harris and I M Smith

March 2003

Testing Algorithms in Standards and METROS

R M Barker, M G Cox, P M Harris and I M Smith
Centre for Mathematics and Scientific Computing

March 2003

ABSTRACT

Many mathematical and statistical problems that arise in metrology can be posed in such a way that they possess a unique correct solution. However, algorithms in standards or software libraries, such as METROS, may only provide an approximate solution or the solution to a nearby problem. An approximate or nearby problem is often introduced so as to give a problem that is (more) tractable, i.e., a problem that *can* be solved or is *easier* to solve and/or is *quicker* to solve. Choosing to solve an approximate problem is one reason why software may not deliver correct results. Another reason is that the implementation of software for solving the approximate problem is poor or faulty. *Algorithm testing* is concerned with understanding the effect of solving an approximate problem by measuring the departure of the solution so obtained from the solution to the correct problem.

A framework is presented for describing the activities of *algorithm testing* and *numerical software testing*, and how these activities contribute to the (overall) validation of the software implementation of an algorithm for achieving a given computational aim. A number of approaches are indicated, based on the methodologies of numerical software testing, for undertaking algorithm testing. The application of the methodologies to a particular case study, viz., the problem of fitting a Gaussian peak function to measurement data, is discussed. The application of the methodologies to additional case studies will be the subject of future work.

ISSN 1471–0005

Authorised by Dr Dave Rayner,
Head of the Centre for Mathematics and Scientific Computing

National Physical Laboratory,
Queens Road, Teddington, Middlesex, United Kingdom TW11 0LW

Contents

1 Introduction

There are a number of specification standards and guides in metrology that require for their implementation certain mathematical and computational analyses to be undertaken [19]. Some of these, e.g., the evaluation of uncertainty in measurement [4], are generic whereas others, e.g., the evaluation of surface texture parameters [1], are aimed at particular metrology areas. Although not always specified in standards there are many other computations that are fundamental to metrology, and many of these will form the basis of software referenced by METROS, a web-based repository of software for solving metrology problems [3].

It is recognised that the *numerical correctness* of software used in metrology is of great concern: a poor software implementation can lead to inaccuracy that could have been avoided, and as a consequence the accuracy of measurement results is compromised. Furthermore, as the accuracy of the hardware components of measurement systems and instruments improves, it becomes increasingly important to understand and quantify the correctness of the software components of those systems. To this end, the National Physical Laboratory, as part of the Software Support for Metrology (SS*f*M) programmes[1], has undertaken work to promote methodologies for testing the numerical correctness of software [5, 13], and to apply these methodologies to software widely used in metrology [8, 9].

Many mathematical and statistical problems that arise in metrology can be posed in such a way that they possess a unique correct solution. However, algorithms in standards or software libraries, such as METROS, may only provide an approximate solution or the solution to a nearby problem. An approximate or nearby problem is often introduced so as to give a problem that is (more) tractable, i.e., a problem that *can* be solved or is *easier* to solve and/or is *quicker* to solve. Choosing to solve an approximate problem is one reason why software may not deliver correct results. Another reason is that the implementation of software for solving the approximate problem is poor or faulty.

Algorithm testing, the subject of this work, is concerned with understanding the effect of solving an approximate problem by measuring the departure of the solution so obtained from the solution to the correct problem. In particular, the aim is to propose approaches to testing the *fitness for purpose* of algorithms used in metrology, and to illustrate the application of these approaches using a number of case studies. The emphasis on fitness for purpose is an important one. It is reasonable to ignore the error introduced by solving an approximate problem if that error is negligible compared with other contributions to the uncertainty associated with the measurement result, e.g., inexact knowledge about the state of the measurement system or instrument, the environment, etc. On the other hand, account must be taken of the approximation error in cases where it is appreciable

[1] See http://www.npl.co.uk/ssfm

compared with these other contributions.

In Section 2 of this report we present a framework for describing the activities of *algorithm testing* and *numerical software testing*, and how these activities contribute to the (overall) validation of the software implementation of an algorithm for achieving a given computational aim. In Sections 3 and 4 we give examples of, respectively, generic and metrology-specific computations to which we might apply the activities of algorithm testing and numerical software testing. Much metrology software is likely to incorporate aspects such as those illustrated using these examples. In Section 5 we indicate a number of approaches, based on the methodologies of numerical software testing, for undertaking algorithm testing. The application of the methodologies to a particular case study, viz., the problem of fitting a Gaussian peak function to measurement data, is discussed in Section 6. It is planned to apply the methodologies described here to further case studies in the future. Conclusions are given in Section 7.

2 A Framework for Software and Algorithm Testing

2.1 Software testing

The development of mathematical software may be described as a process with an input and output as follows:

Input: computational aim C_0

Output: software implementation of an algorithm[2] for solving the problem specified by C_0

The computational aim C_0 is itself described as a mapping from input data $\mathbf{x}$ to output data $\mathbf{y}$, with the mapping taking the form, for example, of a *mathematical* rule or procedure $\mathbf{f}$, i.e.,

$$\mathbf{y} = \mathbf{f}(\mathbf{x}). \tag{1}$$

The development of mathematical software relies on the computational aim, and hence the objects $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{f}$, being specified unambiguously. Some examples of computational aims encountered in metrology are given in Sections 3 and 4.

In practice, what is implemented as software is a *numerical* rule or procedure $\mathbf{f}_a$ which gives output data $\mathbf{y}_a$ corresponding to input data $\mathbf{x}$, i.e.,

$$\mathbf{y}_a = \mathbf{f}_a(\mathbf{x}). \tag{2}$$

[2] A logical arithmetical or computational procedure.

Note that even if $\mathbf{f}_a$ has the same mathematical form as $\mathbf{f}$, it describes here a different "object". This is because $\mathbf{f}$ involves mathematical operations implemented using *infinite* precision arithmetic whereas $\mathbf{f}_a$ involves finite precision arithmetic operations and delivers a finite precision result. *Software testing* is concerned with investigating for a specific item of (test) software the numerical correctness of $\mathbf{y}_a$, i.e., its "closeness" to $\mathbf{y}$. Typically, it is undertaken by:

1. constructing reference pairs (reference input data $\mathbf{x}$ and corresponding reference output results $\mathbf{y}$) corresponding to the computational aim C_0,
2. applying the (test) software to the reference data to obtain test results $\mathbf{y}_a$, and
3. comparing the test results with the reference results, taking into account such factors as the *degree of difficulty* of the computational aim C_0 for the reference data set $\mathbf{x}$ and the machine precision of the arithmetic used for the computation.

For step 1 we use either reference software or data generator software [5, 13].

For step 3 we use a performance measure, such as

$$P = \log_{10}\left(1 + \frac{d}{K_0\eta}\right), \tag{3}$$

where d is the relative error in the test results, K_0 is the degree of difficulty of the computational aim C_0 for the reference data set $\mathbf{x}$, and η is the computational precision[3]. K_0 provides a baseline for comparing the test and reference results. It measures the relative change in the output data $\mathbf{y}$ corresponding to a relative change (typically of the order of η) in the input data $\mathbf{x}$. It reflects the performance of software implementing (correctly) an optimal algorithm applied to the reference data, and accounts for the condition of the computational aim C_0 as well as that associated with the data generation process [5, 8, 9].

For example, suppose for a computational aim C_0 and a reference data set $\mathbf{x}$ that $K_0 = 10000 = 10^4$. This means that, in terms of calculations performed using an arithmetic with computational precision $\eta = 10^{-16}$, the relative error in the result $\mathbf{y}$ can be expected to be as large as $K_0\eta = 10^{-12}$ no matter how good are the algorithm chosen to solve the problem specified by C_0 and the software implementation of that algorithm. Now suppose particular software for C_0 returns results with a relative error of $d = 10^{-8}$. Then, $d/(K_0\eta) = 10^4$, and $P \approx 4$. This indicates that the software is losing *four* significant figures of numerical accuracy over and above those that could be lost by (the software implementation of) an ideal

[3] For the very commonly used floating-point arithmetic, η is the smallest positive representable number u such that the value $1 + u$, computed using the arithmetic, exceeds unity. For the many floating-point processors which today employ IEEE arithmetic, $\eta = 2^{-52} \approx 2 \times 10^{-16}$, corresponding to approximately 16-digit working.

Required computational aim C_0	$\rightarrow$	Approximate computational aim C_a
$\downarrow$		$\downarrow$
Algorithm A_0 to solve the problem specified by C_0		Algorithm A_a to solve the problem specified by C_a
$\downarrow$		$\downarrow$
Software S_0 implementing A_0		**Software** S_a implementing A_a

Figure 1: Objects in the process of developing mathematical software: computational aims, algorithms and software.

algorithm. A value for P of approximately zero would indicate that the software is losing a number of figures of numerical accuracy that is *comparable* to the number that could be expected to be lost by (the software implementation of) an ideal algorithm.

The testing described above only makes use of knowledge of the computational aim C_0, and treats the software as a "black-box". In practice, however, software may fail to deliver results having sufficient accuracy for the application of concern for one or both of the following reasons:

1. The software addresses a computational aim C_a that approximates C_0.
2. The implementation of software to solve the problem specified by C_a is poor or faulty.

An approximate computational aim C_a is often introduced to give a problem that is easier and/or quicker to solve compared with the problem specified by C_0. Some examples are given in Sections 3 and 4. In the testing undertaken to date [5, 8, 9] we have not explicitly been concerned with distinguishing between these two reasons for software failing to deliver correct results, although examples have been documented [6] where it has been possible to attribute incorrect results to one or other cause.

2.2 Concepts

There are different objects that we are concerned with and may wish to compare. We have computational aims (mathematics), algorithms (pseudo-code) and software (source or object code): see Figure 1.

In extreme cases only the objects shown in bold in Figure 1 may exist: we may not know of an algorithm (or software implementation) to solve the problem specified by C_0. Equally we may not be given the algorithm (or the intended computational aim) used by the software S_a.

We can ask questions about objects of a different type (in the same column of Figure 1):

- To what extent does an algorithm solve the problem specified by its given computational aim?
- To what extent does a software implementation follow its given algorithm?
- To what extent does a software implementation solve the problem specified by its given computational aim?

We can make comparisons between objects of the same type (in different columns of Figure 1):

- To what extent are two computational aims equivalent?
- To what extent are two algorithms (for solving problems specified by the same computational aim) equivalent?
- To what extent do two software implementations (of the same algorithm) give the same answer?

We can make mathematical and numerical comparisons between the various objects in Figure 1:

- Two computational aims and the algorithms for solving the problems specified by them (or two algorithms for solving problems specified by the same computational aim) are *mathematically equivalent* for a given set of input data if the algorithms deliver identical results for that data when implemented correctly using *infinite* precision arithmetic.
- Two algorithms and the software implementing them (for solving problems specified by the same computational aim) are *numerically equivalent* for a given set of input data if the software delivers results for that data accurate to within a stated bound (dependent on the problem condition, the input data and the computer arithmetic) when implemented correctly using *finite* precision arithmetic.

Some of the comparisons can be undertaken using software testing techniques. In particular, any comparison involving software can be made this way. We assume,

therefore, that we can test (using software testing techniques) that the software S_a is an implementation of the approximate algorithm A_a and solves the problem specified by the approximate computational aim C_a. Furthermore, if the computational aim addressed by the test software is not specified, we assume we can deduce it, and then use software testing techniques to verify our deductions.

2.3 Software testing revisited

Software testing is concerned with testing the accuracy to which the software S_a delivers results for the computational aim C_0. In terms of the objects introduced in Section 2.2 it can be described in terms of two activities:

Algorithm testing investigating the extent of the mathematical equivalence of the computational aims C_0 and C_a

Numerical software testing investigating the numerical accuracy to which the software S_a solves the problem specified by its computational aim C_a.

These activities relate very clearly to the two reasons enumerated in Section 2.1 for software failing to deliver correct results. The focus of the work reported here is on the first of these activities.

2.4 Fitness for purpose

Generally, the input data $\mathbf{x}$ will contain values of

- measurements obtained from an experiment,
- corrections and correction factors to be applied to the measurements, and
- physical constants (such as material constants),

and the output data $\mathbf{y}$ will be the corresponding value of a desired measurement result. In practice, however, the input data is not known exactly and if the experiment were to be repeated we would expect to record different input data values and, consequently, to obtain different values for the measurement result. Understanding the inexactness or *uncertainty* associated with the measurement result arising from uncertainty associated with the input data is a key activity in metrology, underpinning our ability to judge the consistency between experiment and theory, between different measurements and between different laboratories.[4]

[4] Account also needs to be taken of systematic errors associated with the measurement result. Here, the concentration is on "precision" as opposed to "trueness" [2].

Suppose $\mathbf{X}$ and $\mathbf{Y}$ denote the input and output quantities, regarded here as *random variables*, and related by the model

$$\mathbf{Y} = \mathbf{f}(\mathbf{X}) \tag{4}$$

(cf. equation (1)) that defines the required computational aim C_0. The inexactness of the input and output quantities is described by the (joint) *probability density functions* $\mathbf{g}(\mathbf{X})$ and $\mathbf{g}(\mathbf{Y})$, respectively. The problem addressed in uncertainty evaluation [4] is to determine $\mathbf{g}(\mathbf{Y})$ from $\mathbf{f}$ and the available knowledge about $\mathbf{g}(\mathbf{X})$.[5]

As we have seen, in practice the computation of the measurement result is undertaken using software that solves a problem specified by an approximate computational aim C_a, i.e., (4) takes the form

$$\mathbf{Y}_a = \mathbf{f}_a(\mathbf{X}) \tag{5}$$

(cf. equation (2)). Then, the computed measurement result will have its own (joint) probability density function $\mathbf{g}(\mathbf{Y}_a)$ determined from $\mathbf{f}_a$ and the same knowledge about $\mathbf{g}(\mathbf{X})$.

Let us suppose there is a single (univariate) output quantity[6], i.e., (4) and (5) take the form

$$Y = f(\mathbf{X})$$

and

$$Y_a = f_a(\mathbf{X}),$$

and the probability density functions for the required and computed measurement results are, respectively, $g(Y)$ and $g(Y_a)$. We require measures of the departure of C_a from C_0 that can be interpreted in terms of the metrology problem solved. In particular, we require to assess the *fitness for purpose* of solving C_a in place of C_0 for given input data $\mathbf{x}$ accounting for the inexactness of that input data.

There are a number of questions we might ask:

1. What is the departure of the computed measurement result y_a from the required measurement result y?

2. What is the probability of obtaining a solution to the problem specified by the required computational aim C_0 that is further from y than is y_a?

3. "On average" what is the departure of y_a from y?

[5] The available knowledge typically includes (a) best estimates of $\mathbf{X}$ (given by the input data values $\mathbf{x}$), and (b) information about the dispersion of values that could reasonably be attributed to $\mathbf{X}$.

[6] The generalisation to a multivariate measurement result is straightforward.

To answer the first question, we can use measures of the form

$$|y_a - y|$$

and

$$\frac{|y_a - y|}{|y|}, \quad y \neq 0,$$

which quantify, respectively, the absolute and relative departures of y_a from y but tell us nothing about the fitness for purpose of y_a in the sense described above. In some cases it will be possible to derive *analytic* expressions for these measures, viz., *error bounds* (Section 5.1). Otherwise, it is necessary to evaluate the measures using values for y_a and y calculated using software for solving the problems specified by C_a and C_0 (Section 5.2).

Note that the objective here is to measure the departure of the solutions to the problems specified by C_a from C_0. However, the departure determined in terms of values y_a and y calculated using *software* for the two computational aims will additionally include contributions relating to the numerical accuracy of those calculated values. Numerical software testing techniques enable us to understand and quantify these contributions. It may be safe to ignore the contributions if the errors introduced by the software are small compared to (a) the calculated departure (Sections 5.2 and 5.3), or (b) the inexactness of the required and computed measurement results described by, respectively, the probability density functions $g(Y)$ and $g(Y_a)$ (Section 5.4).

To answer the second question, we wish to evaluate

$$Pr\left(|Y - y| \geq |y_a - y|\right),$$

where "$Pr(A)$" denotes "probability of A occurring". This corresponds to finding the area under the curve $g(Y)$ to the left of $y - |y_a - y|$ and to the right of $y + |y_a - y|$ (see Figure 2), and can therefore be evaluated given knowledge of $g(Y)$. A small probability suggests that y_a, the solution obtained for the approximate computational aim C_a, is *not* a likely solution to C_0 accounting for the inexactness of the input data values.

To answer the third question, we wish to compare the probability density functions $g(Y)$ with $g(Y_a)$ (see Figure 3). Particular measures of interest are the differences between the expectation values of the random variables Y and Y_a and between the variances of those random variables. These measures provide, respectively, information about the *bias* and *efficiency* of the approximate computational aim with respect to the required computational aim. In particular, the difference between the expectation values provides information about the average deviation between solutions to the two computational aims. In the example illustrated in Figure 3 we may conclude that (a) there is an appreciable bias between the solutions to the two computational aims, and (b) the dispersion of values of possible solutions to the

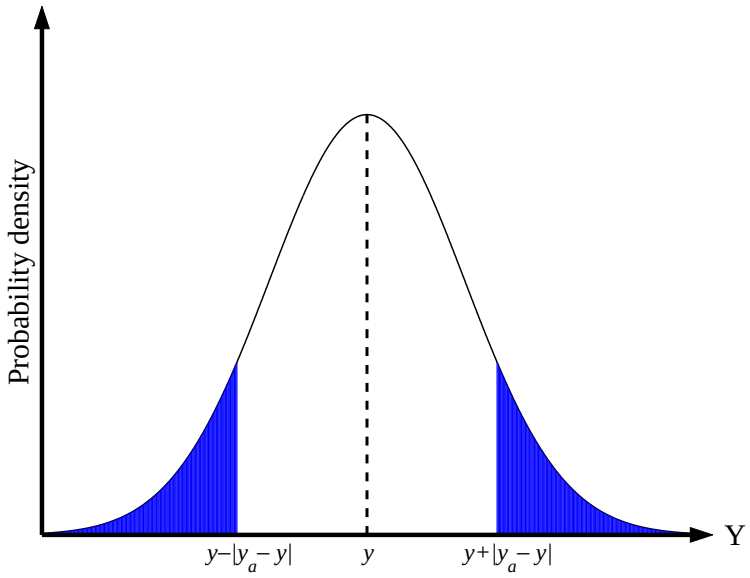

Figure 2: A measure of the fitness for purpose of solving C_a in place of C_0: the area of the shaded regions represents the probability of obtaining a solution to C_0 that is further from y than is y_a.

---

problem specified by the approximate computational is appreciably greater than that for the required computational aim.

3 Generic Examples

3.1 Sample variance

The computational aim C_0 is to find the sample variance s^2 of the data $\{x_i : i = 1, \ldots, m\}$.

Software S_0^A solves the problem specified by C_0 by implementing the algorithm

$$s^2 = \frac{1}{m-1} \sum_{i=1}^{m} (x_i - \bar{x})^2 \,, \tag{6}$$

where

$$\bar{x} = \frac{1}{m} \sum_{i=1}^{m} x_i$$

is the sample mean.

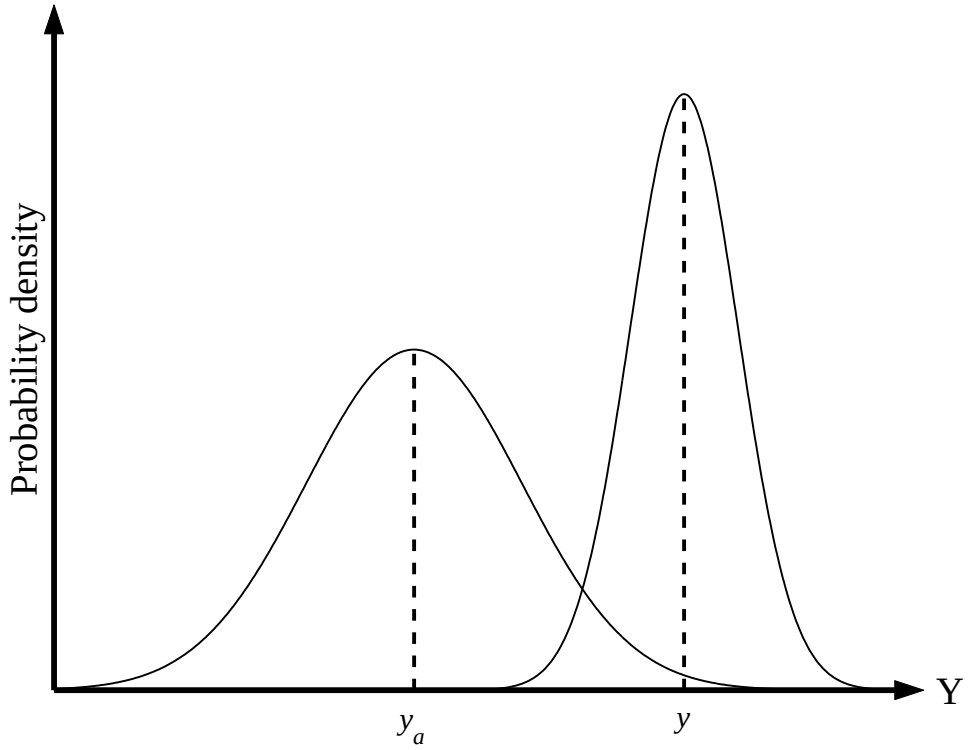

Figure 3: Probability density functions $g(Y)$ and $g(Y_a)$ for, respectively, the required and computed measurement result.

Software S_0^B solves the problem specified by C_0 by implementing the algorithm

$$s^2 = \frac{1}{m-1}\left\{\sum_{i=1}^{m} x_i^2 - \frac{1}{m}\left(\sum_{i=1}^{m} x_i\right)^2\right\}. \tag{7}$$

The algorithms are mathematically equivalent but numerically different (Section 2.2). If implemented correctly using infinite precision arithmetic, the two formulae (6) and (7) would deliver identical values for the sample variance when applied to identical data. However, if implemented correctly using finite precision arithmetic S_0^B will generally deliver results that are (numerically) less accurate than those delivered by S_0^A for the same data. Numerical software testing can be used to distinguish between the two software implementations [5, 9].

3.2 Linear regression

The computational aim C_0 is to solve for $\mathbf{y}$ the overdetermined system of linear equations

$$A\mathbf{y} = \mathbf{x}$$

in a least-squares sense.

Software S_0^A solves the problem specified by C_0 by forming and solving the normal equations [12, section 4]:

$$\left(A^{\mathrm{T}}A\right)\mathbf{y} = A^{\mathrm{T}}\mathbf{x}. \tag{8}$$

Software S_0^B solves the problem specified by C_0 using an orthogonal matrix factorisation of A [12, section 4]:

$$A = QR, \quad R\mathbf{y} = Q^{\mathrm{T}}\mathbf{x}, \tag{9}$$

where Q is an orthogonal matrix ($Q^{\mathrm{T}}Q = I$, the identity matrix) and R is upper-triangular.

The algorithms are mathematically equivalent but numerically different. Numerical software testing can be used to distinguish between the two software implementations, and to show that S_0^A delivers results that can be numerically much less accurate than those delivered by S_0^B for the same data.

3.3 Gaussian peak fitting

The computational aim C_0 is to fit in a least-squares sense a Gaussian peak model [6],

$$y(x) = A\exp\left(-\frac{(x-\bar{x})^2}{2s^2}\right) \equiv \exp\left(a_0 + a_1 x + a_2 x^2\right), \quad a_2 < 0, \tag{10}$$

to measurement data $\{(x_i, y_i) : i = 1, \ldots, m\}$, i.e., to solve the problem

$$\min_{A,\bar{x},s} \sum_{i=1}^{m} \left\{ y_i - A \exp\left(-\frac{(x - \bar{x})^2}{2s^2} \right) \right\}^2 .$$

Software S_1 solves the problem specified by C_1:

$$\min_{\mathbf{a}} \sum_{i=1}^{m} \left\{ y_i - \exp\left(a_0 + a_1 x_i + a_2 x_i^2 \right) \right\}^2 . \tag{11}$$

Software S_2 solves the problem specified by C_2:

$$\min_{\mathbf{a}} \sum_{i \in I} \left\{ \ln y_i - \left(a_0 + a_1 x_i + a_2 x_i^2 \right) \right\}^2 , \tag{12}$$

where

$$I = \{ i : y_i > 0 \} .$$

The computational aims C_0, C_1 and C_2 are not mathematically equivalent. C_0 and C_1 are not equivalent because a solution to C_1 may be such that $a_2 > 0$ which does not correspond to a Gaussian peak model as specified in (10). C_1 and C_2 are not mathematically equivalent because they involve minimising different objective functions. Moreover, if there exists an index i for which $y_i \leq 0$, the summations in (11) and (12) will involve different subsets of the data. Algorithm testing is used to investigate the effect of using C_1 or C_2 in place of C_0, assuming correct software implementations of algorithms for those computational aims. Numerical software testing is used to investigate particular software implementations of algorithms for solving, respectively, the problems specified by C_1 and C_2.

Now consider particular ways of solving the problem specified by C_1:

- Software S_1^A undertakes the minimisation (11) using a Gauss-Newton algorithm that iterates until there is no further reduction in the sum of squares objective function
- Software S_1^B undertakes the minimisation (11) using a Gauss-Newton algorithm that iterates until the change in the value of the sum of squares objective function is less than a prescribed tolerance
- Software S_1^C undertakes the minimisation (11) using a Gauss-Newton algorithm that iterates until the changes in the estimates of a_0, a_1 and a_2 are less than a prescribed tolerance

The software S_1^A, S_1^B and S_1^C are solving problems that are not mathematically equivalent. This is because, even when implemented correctly using infinite precision arithmetic, they will generally return different solutions. Denote by C_1^A, C_1^B and C_1^C the computational aims addressed by, respectively, S_1^A, S_1^B and S_1^C. Algorithm testing is used to investigate the effect of using C_1^A, C_1^B or C_1^C in place of C_0. Numerical software testing is used to investigate whether test software S_1^A, S_1^B and S_1^C implement correctly a Gauss-Newton algorithm with their respective termination criteria.

3.4 Uncertainty evaluation

The computational aim C_0 is to determine the probability density function $g(Y)$ of the random variable $Y = f(\mathbf{X})$ given the measurement model f and the (joint) probability density function $\mathbf{g}(\mathbf{X})$ of the random variable $\mathbf{X}$ [11].

Software S_1 solves the problem specified by C_1: apply the law of propagation of uncertainty (based on a linearisation of the model) and the Central Limit Theorem [4].

Software S_2 solves the problem specified by C_2: apply Monte Carlo simulation as an implementation of the principle of propagation of distributions [7, 10, 11].

The computational problems C_0, C_1 and C_2 are not mathematically equivalent. The probability density function $g(Y)$ cannot generally be expressed in simple or even closed mathematical form. Formally, the computational problem C_0 is to evaluate

$$g(\eta) = \int_{\xi \in \mathcal{R}^n} \mathbf{g}(\xi)\delta(\eta - f(\xi))d\xi,$$

where $\delta(\cdot)$ denotes the Dirac delta function [11]. C_1 assigns a Gaussian distribution[7] for $g(Y)$ (irrespective of the "correct" distribution) with a standard deviation computed on the basis of a linearisation of the measurement model. C_2 assigns an empirical estimate of the "correct" distribution, the accuracy of which depends on a number of factors including the properties of the pseudorandom number generators required by the Monte Carlo calculation and the number of Monte Carlo trials undertaken. Algorithm testing is used to investigate the effect of using C_1 or C_2 in place of C_0, assuming correct software implementations of algorithms for solving the problems specified by those computational aims. Numerical software testing is used to investigate particular software implementations of algorithms for solving, respectively, the problems specified by C_1 and C_2.

This is a computation specified in a guide to measurement uncertainty [4].

[7] In the case that the effective degrees of freedom associated with the measurement result is finite, a (scaled and shifted) t distribution is assigned instead of a Gaussian distribution [4].

4 Metrology-Specific Examples

4.1 Calculation of conventional mass values

The conventional (reported) mass m_0 of a weight is defined as the mass of a (hypothetical) weight of density $\rho_0 = 8000$ kg/m^3 that balances the weight of mass m and density ρ in air at density $a_0 = 1.2$ kg/m^3:

$$m_0 \left(1 - \frac{a_0}{\rho_0}\right) = m \left(1 - \frac{a_0}{\rho}\right)$$

from which

$$m_0 = m \left(1 - \frac{a_0}{\rho}\right) \left(1 - \frac{a_0}{\rho_0}\right)^{-1} \approx m \left(1 - a_0 \left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)\right).$$

The required computational aim C_0 is to evaluate m_0 from

$$m_0 = m \left(1 - \frac{a_0}{\rho}\right) \left(1 - \frac{a_0}{\rho_0}\right)^{-1}. \tag{13}$$

The approximate computational aim C_a is to evaluate $m_{0,a}$ from

$$m_{0,a} = m \left(1 - a_0 \left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)\right). \tag{14}$$

Algorithm testing is used to investigate the effect of using C_a in place of C_0.

This is a key function in METROS [3].

4.2 Surface texture analysis

There are many types of instrument for the measurement of surface texture. The most direct are instruments in which some form of probe is translated across the surface either in contact with it or at some fixed distance from it [20]. The vertical displacement of the probe is recorded as a function of the horizontal displacement to provide an indication of the profile of the surface. Although it is possible to provide calibration and traceability for the measurement of vertical and horizontal displacement and the orthogonality of the vertical and horizontal axes, this does not itself provide a calibration of the profile of the surface because there will inevitably be some interaction between the surface and the probe.

Consider the example of a surface whose cross-sectional profile is a periodic function and the measurement of that profile by an instrument with a probe that contacts the surface. The data recorded by the instrument consists of the locus of the

probe-tip centre. For sufficiently small probe-tip radii, i.e., less than the smallest radius of curvature of the surface profile, this locus represents an *offset* curve. An offset curve is a curve that is a constant distance from the surface profile, where distance is measured orthogonally to the profile. In particular, the amplitude of the offset curve for a periodic function will be equal to that of the periodic function itself when the above curvature condition applies. This statement is true because the probe tip will make physical contact with the surface profile at the peaks and troughs of the latter in this case. Thus, an estimate A of the amplitude of the surface profile is given by that of the offset curve. Moreover, an estimate L of the wavelength of the fundamental frequency can similarly be determined from the offset curve.

In order to describe the *shape* of the surface profile $p(x)$, it is represented by the n-harmonic Fourier series, viz.,

$$p(x) = a_0 + \sum_{k=1}^{n} a_k \cos \frac{2\pi kx}{L} + b_k \sin \frac{2\pi kx}{L}. \tag{15}$$

However, the representation of the *offset* curve in the form (15) does *not* provide directly the coefficients a_k and b_k in the above expression, and hence information about the *shape* of the surface profile, because the offset curve for a pure sinusoid is not a pure sinusoid.

The ability to separate the effects of the probe-surface interaction from the ("true") shape of the surface profile is central to the problem of *calibrating* the surface. It is also important to subsequent understanding of the measurements made by a different instrument of the surface profile where the surface is to be used as a *transfer standard*.

The computational aim C_0 is to determine estimates for the coefficients (and their associated uncertainties) in the representation (15) of the surface profile from measurement data representing points on an offset curve recorded by an instrument with a probe of known radius. An approximate computational aim C_a is to use the instrument to measure the topside and underside of the surface profile, recording data representing points on two offset curves, one above and one below the surface profile. *Estimates* of the required coefficients are then provided by the n-harmonic Fourier series representation of the curve *midway* between the two offset curves.

It can be expected that for "sufficiently small" probe-tip radii, a solution to C_a will provide an adequate approximation to a solution to the problem specified by C_0. Here, the meaning of *sufficiently small* will depend on the measured surface profile (specified, e.g., by nominal values for A and L). Moreover, deciding whether the approximation is *adequate* requires us to quantify the uncertainty associated with a solution to the problem specified by C_0, which is likely to depend on the measured surface profile as well as the measurement capabilities of the instrument (including the calibration of its measurement of vertical and horizontal displace-

ments). Algorithm testing is used to investigate the circumstances, relating to the instrument, its probe and the measured surface profile, for which it is acceptable to solve the problem specified by the approximate computational aim C_a in place of the required computational aim C_0.

4.3 Calibration of pressure balances

An important step in the calibration of pressure balances is the estimation of the effective area of the piston-cylinder assembly. At temperature t with applied mass m (corrected for air buoyancy), a pressure balance generates a pressure p given implicitly by

$$p = \frac{(m+c)g}{A(p,\mathbf{a})(1+\phi(t))}, \tag{16}$$

where ϕ is a known function of temperature t ($|\phi(t)| \ll 1$) that accounts for a temperature correction, c is a measured constant obtained from a precise characterization of the balance, $A(p,\mathbf{a})$ describes the effective area of the balance in terms of the pressure p and calibration parameters $\mathbf{a}$, and g is gravitational acceleration. In practice, $A(p,\mathbf{a})$ is sufficiently approximated by a_1+a_2p with $a_1 = A_0$, $a_2 = A_0\lambda$, and a_2p small compared to a_1. Consequently, (16) takes the form

$$p = \frac{(m+c)g}{(a_1+a_2p)(1+\phi(t))}. \tag{17}$$

Given a set of measurements p_i, m_i and t_i of pressure, applied load and temperature, and knowledge of c, calibrating a pressure balance means finding values for the parameters $\mathbf{a}$ that best-fit the model (17) to the measurement data.

In a cross-float experiment [17, 18], the pressure balance to be calibrated is connected to a reference balance whose effective area parameters $\mathbf{b}$ have previously been determined, and the applied loads on the two balances are adjusted so that both are in pressure and flow equilibrium. Suppose the reference balance generates a pressure given by (17) with m, c, $\mathbf{a}$ and $\phi(t)$ replaced by M, C, $\mathbf{b}$ and $\Phi(T)$, respectively. Then, when the balances are in equilibrium, we have

$$\frac{(m+c)g}{(a_1+a_2p)(1+\phi(t))} = \frac{(M+C)g}{(b_1+b_2p)(1+\Phi(T))}, \tag{18}$$

where the pressure p is estimated from the calibration of the reference balance.

We consider three computational aims for determining the best fit parameters $\mathbf{a}$ to measurement data [14].

C_0**:** The weighted linear least squares estimator (WLLS) determines estimates of the parameters $\mathbf{a}$ by solving

$$\min_{\mathbf{a}} \sum_i w_i^2 f_i^2(\mathbf{a}),$$

where

$$f_i(\mathbf{a}) = \frac{(m_i + c)g}{1 + \phi(t_i)} - a_1 p_i - a_2 p_i^2,$$

a linear function of $\mathbf{a} = (a_1, a_2)^{\mathrm{T}}$, is the error in satisfying (17), and w_i is inversely proportional to the standard uncertainty associated with f_i.

C_1**:** The P-estimator (PLLS) determines parameter estimates by solving

$$\min_{\mathbf{a}} \sum_i e_i^2(\mathbf{a}),$$

where

$$e_i(\mathbf{a}) = \left[\frac{m_i + c}{M_i + C}\right]\left[\frac{1 + \Phi(T_i)}{1 + \phi(t_i)}\right] - \frac{a_1 + a_2 p_i}{b_1 + b_2 p_i},$$

is the error in satisfying (18).

C_2**:** The ΔP-method (DLLS) [16] determines parameter estimates by solving

$$\min_{\mathbf{a}} \sum_{i>1} d_i^2(\mathbf{a}),$$

where

$$d_i(\mathbf{a}) = \frac{(m_i - m_1)g}{[p_i - p_1 + (\phi(t_i) - \phi(t_1)p_1][1 + \phi(t_i)]} - a_1 - a_2(p_i + p_1).$$

Algorithm testing is used to investigate the effect of using C_1 or C_2 in place of C_0 assuming correct software implementations of algorithms for these computational aims. Numerical software testing is used to investigate particular software implementations of algorithms for solving the problems specified by C_0, C_1 and C_2.

Note that our definition of C_0 above as the required computational aim follows from the assumptions we are prepared to make about the measurement data, in particular that the (significant) errors in that data are associated only with the measurements of the masses $m_i,\ i = 1, \ldots, m$, loaded on the balance to be calibrated. If we wish to account for possible errors in *other* measurements (e.g., the pressures p_i derived from the reference balance, the temperatures t_i, etc.), it would be appropriate to change the definition of the computational aim C_0 to reflect the new set of assumptions. Therefore, the required computational aim C_0 can be expected to change as our knowledge of the metrology problem improves. In such circumstances, algorithm testing may be used to investigate whether an approximate computational aim C_a *continues* to provide an adequate approximation to C_0 (supplemented by the new knowledge).

5 Methods for Algorithm Testing

5.1 Methods based on the analysis of computational aims

Analysis of the required and approximate computational aims to understand the circumstances under which they are mathematically equivalent, and to derive expressions for the error between the solutions to the problems specified by them, is an important approach to algorithm testing. The approach involves working with the computational aims, and the algorithms for solving the problems specified by them, as *mathematical objects* independent of any software implementation. However, the approach has the disadvantages of generally requiring a high degree of technical skill (and therefore relying on experts), being highly problem-specific, and consequently being difficult (if not impossible) to automate.

Nevertheless, the use of analysis in some of the examples described in Sections 3 and 4 leads to the following results:

- For the calculation of the sample variance s^2 (Section 3.1), we can use the mathematical rules of algebra to show that the algorithms (6) and (7) are mathematically equivalent. Similarly, the algorithms (8) and (9) for solving the linear regression problem (Section 3.2) are mathematically equivalent.

- For the problem of uncertainty evaluation (Section 3.4) we can state the conditions under which the application of the law of propagation of uncertainty and the Central Limit Theorem solves *correctly* the problem specified by the required computational aim of finding the probability density function for the value of $Y = f(\mathbf{X})$. These conditions are that the (joint) probability density function for $\mathbf{X}$ is a multivariate Gaussian and the model f is a linear function of the quantities $\mathbf{X}$. Furthermore, we can state that the use of Monte Carlo simulation as an implementation of the principle of the propagation of distributions will *always* provide an approximate solution to the problem specified by the required computational aim, but with a degree of accuracy (measured in a statistical sense) that is a function of the number of Monte Carlo trials (and hence under the user's control), and also to some extent a function of the properties of the pseudorandom number generators used[8].

- For the calculation of conventional mass values (Section 4.1), we can derive an expression for the *error* between the required and approximate values returned by the algorithms (13) and (14), viz., applying Taylor's theorem,

$$m_0 \;=\; m\left(1 - \frac{a_0}{\rho}\right)\left(1 - \frac{a_0}{\rho_0}\right)^{-1}$$

[8] Good generators are available [10], and their use means that their influence is minimal.

$$
\begin{aligned}
&= m\left(1-\frac{a_0}{\rho}\right)\left(1+\frac{a_0}{\rho_0}+\left(\frac{a_0}{\rho_0}\right)^2\left(1-\theta\frac{a_0}{\rho_0}\right)^{-3}\right)\\
&= m_{0,a}+m\left(-\frac{a_0^2}{\rho\rho_0}+\left(\frac{a_0}{\rho_0}\right)^2\left(1-\frac{a_0}{\rho}\right)\left(1-\theta\frac{a_0}{\rho_0}\right)^{-3}\right),
\end{aligned}
$$

for some θ satisfying $0 < \theta < 1$. This allows us to bound the error associated with calculating an approximate conventional mass value and, by comparing this with the uncertainty associated with the required value, judge the fitness for purpose of the approximate value.

- For the problem of calibrating pressure balances (Section 4.3), we can state the circumstances under which the approximate computational aims defined by the P-estimator and ΔP-method are mathematically equivalent to the required computational aim. For example [14], the P-estimator is equivalent to the required computational aim in the case that the weights w_i in the weighted linear least-squares problem defining the required computational aim are set to be inversely proportional to the pressures p_i. This circumstance applies if the only measurements subject to error are those of the masses m_i, and these errors are samples from a probability distribution whose standard deviation is proportional to pressure.

An analysis of the required and computational aims for solving the Gaussian peak fitting problem (Section 3.3) is given in Section 6.

5.2 Methods based on using software implementations of the computational aims

In some circumstances we will have available *software* implementing the required and approximate computational aims. A direct approach to comparing the computational aims using this software is as follows:

1. Start with a reference data set $\mathbf{x}$ (perhaps typical of one that is likely to be encountered in practice).
2. Apply software for the required computational aim C_0 to the reference data set $\mathbf{x}$ to obtain reference results $\mathbf{y}_0$.
3. Apply software for the approximate computational aim C_a to the reference data set $\mathbf{x}$ to obtain test results $\mathbf{y}_a$.
4. Measure the difference between the computational aims C_0 and C_a by the distance (in an appropriate norm) between the reference results $\mathbf{y}_0$ and the test results $\mathbf{y}_a$.

As discussed in Section 2.4, the distance evaluated at Step 4 includes contributions relating to the accuracy of the calculations of the reference and test results, and so may not provide an accurate measurement of the departure of C_a from C_0.

Let $\tilde{\mathbf{y}}_0$ and $\tilde{\mathbf{y}}_a$ denote the *mathematical solutions* to the problems specified by the computational aims and the reference data set $\mathbf{x}$. Then, we may write

$$\tilde{\mathbf{y}}_a - \tilde{\mathbf{y}}_0 = (\tilde{\mathbf{y}}_a - \mathbf{y}_a) + (\mathbf{y}_a - \mathbf{y}_0) + (\mathbf{y}_0 - \tilde{\mathbf{y}}_0).$$

The term on the left-hand-side is the (required) difference between the mathematical solutions to the problems specified by the computational aims. In the above equality, this is expressed in terms of the difference between the calculated results (the second term on the right-hand-side) and the errors in the calculated results returned by the software implementations of the computational aims (the first and last terms on the right-hand-side). We can make the first and last terms as small as we can by using *reference software* for the two computational aims. However, even then the magnitude of these terms will be bounded by terms that reflect the inherent conditioning of the problems specified by the computational aims, i.e.,

$$\|\tilde{\mathbf{y}}_a - \mathbf{y}_a\| \leq K_a \eta \|\mathbf{y}_a\|,$$

and

$$\|\tilde{\mathbf{y}}_0 - \mathbf{y}_0\| \leq K_0 \eta \|\mathbf{y}_0\|,$$

where K_a and K_0 are the degrees of difficulty corresponding to, respectively, C_a and C_0 and the reference data set $\mathbf{x}$. Numerical software testing is used to investigate whether the software used to calculate $\mathbf{y}_a$ and $\mathbf{y}_0$ is performing like reference software (and the above bounds apply) by quantifying the magnitude of the errors in the calculated results. In particular, in the case that the magnitudes of the errors are small compared with $\|\mathbf{y}_a - \mathbf{y}_0\|$, the latter can be used to measure the difference between the computational aims as required.

The application of the approach to the Gaussian peak fitting problem (Section 3.3) is given in Section 6.

5.3 Methods based on data generation software for the computational aims

The generation of *reference pairs*, comprising reference data sets and corresponding reference results, appropriate to a specified computational aim is the basis of "black-box" testing of numerical software [5, 13]. One approach to generating reference pairs is to use reference software (Section 5.2). Another approach is to use a *data generator*. This is software for constructing reference data sets with known solutions, i.e., solutions specified *a priori*, by solving the inverse problem to that defined by the computational aim. For a problem whose solution can be

characterised mathematically, the implementation of a data generator is generally much simpler than the implementation of software for solving the forward problem defined by the computational aim. Data generators have been used in the numerical testing of software for the calculation of the sample mean and sample standard deviation [9] and linear regression [8]. In addition, NPL has undertaken work to develop data generation software for a number of calculations important in metrology, including the determination of associated (geometric) features and tolerance assessment calculations in dimensional metrology.

In the context of algorithm testing, data generation software for the required and approximate computational aims may be used to compare the computational aims in the following manner:

1. Start with a reference solution denoted by $\mathbf{y}$.
2. Use data generation software appropriate to the required computational aim C_0 to generate reference data $\mathbf{x}_0$ corresponding to the reference solution $\mathbf{y}$.
3. Use data generation software appropriate to the approximate computational aim C_a to generate reference data $\mathbf{x}_a$ corresponding to $\mathbf{y}$.
4. Measure the difference between the computational aims C_0 and C_a by the distance (in an appropriate norm) between the data sets $\mathbf{x}_0$ and $\mathbf{x}_a$.

The process of data generation in steps 2 and 3 above is a one-to-many mapping, i.e., in each case there are *many* reference data sets $\mathbf{x}_0$ and $\mathbf{x}_a$ for the two computational aims that have the same reference solution $\mathbf{y}$. Consequently, at step 2 we choose a particular reference data set $\mathbf{x}_0$ that is "close" (in an appropriate norm) to a data set $\mathbf{x}$ that is typical of one likely to be encountered in practice. Then, at step 3 we choose a reference data set $\mathbf{x}_a$ that is "closest to" $\mathbf{x}_0$ (in the same norm).

If the distance between the reference data sets $\mathbf{x}_0$ and $\mathbf{x}_a$ constructed in this way is *zero*, the computational aims are equivalent for a problem specified by the reference pair $(\mathbf{x}, \mathbf{y})$. If the distance is *small* compared to the uncertainty associated with measurements of typical data $\mathbf{x}$, the computational aim C_a may be regarded as fit for purpose. Otherwise, we may question whether the approximate computational aim used in place of the required computational aim is adequate for its intended purpose.

As in Section 5.2 we must be aware that we are using software to calculate the reference data sets $\mathbf{x}_0$ and $\mathbf{x}_a$ and, consequently, the distance between the data sets includes contributions relating to the accuracies of those data sets. However, as stated, often the computations undertaken by the data generators are much simpler than those required to solve the problems specified by the computational aims, and so this aspect is often of no concern.

The approach is illustrated in Figure 4). In this figure $D_0(\mathbf{y})$ represents the collection of data sets $\mathbf{x}_0$ for which the solution to the problem specified by the computational aim C_0 is $\mathbf{y}$. Similarly, $D_a(\mathbf{y})$ represents the collection of data sets $\mathbf{x}_a$ for which the solution to the problem specified by the approximate computational aim C_a is $\mathbf{y}$. $\mathbf{x}$ is a data set that is typical of the application: then, $\mathbf{x}_0$ is the data set in $D_0(y)$ closest to $\mathbf{x}$, and $\mathbf{x}_a$ the data set in $D_a(y)$ closest to $\mathbf{x}_0$. Finally, the distance d between $\mathbf{x}_0$ and $\mathbf{x}_a$ measures the difference between the computational aims for the data set $\mathbf{x}$.

The approach is similar to the technique of *backward error analysis* [21] used to investigate the numerical properties of algorithms implemented using finite precision arithmetic. An advantage of the approach is that software implementing algorithms to solve the problems specified by the computational aims is not required.

The application of the approach to the Gaussian peak fitting problem (Section 3.3) is given in Section 6.

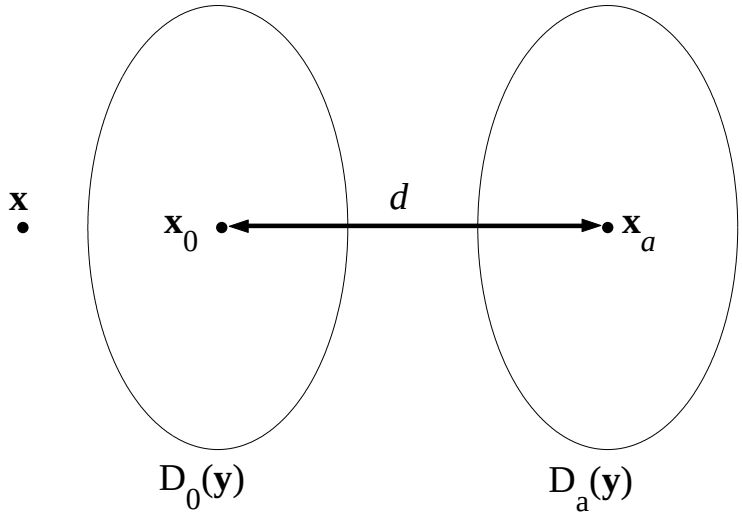

Figure 4: Comparing required and approximate computational aims by the application of data generation software.

5.4 Methods based on simulation

The methods described in the previous sections provide point measures of the departure of C_a from C_0 for the single reference data set $\mathbf{x}$. It is recommended that the methods are applied for a range of reference data sets typical of those likely to be encountered in practice. This can mean sampling (perhaps uniformly) from the domain of all possible data sets that are likely to be encountered, including those close to the boundaries of this domain. In addition, and in order to obtain statistical

measures of the departure and, in particular, to assess the fitness for purpose of C_a as a substitute for C_0 (Section 2.4), it is recommended that the methods are applied using reference data sets sampled from the probability density function (pdf) $\mathbf{g}(\mathbf{X})$ that describes the available knowledge about the inexactness of the input data.

For example, software for solving the problems specified by the two computational aims (Section 5.2) may be combined with simulation in the following manner:

1. Choose a number M of trials.
2. For $r = 1, \ldots, M$:
   (a) Construct the data set $\mathbf{x}^r$ as a random sample from the pdf $\mathbf{g}(\mathbf{X})$.
   (b) Apply software for the required computational aim C_0 to the data set $\mathbf{x}^r$ to obtain reference results $\mathbf{y}_0^r$.
   (c) Apply software for the approximate computational aim C_a to the data set $\mathbf{x}^r$ to obtain test results $\mathbf{y}_a^r$.
3. Construct an estimate $\tilde{\mathbf{g}}(\mathbf{Y})$ of the pdf $\mathbf{g}(\mathbf{Y})$ from the set of reference results $\mathbf{y}_0^r,\ r = 1, \ldots, M$.
4. Construct an estimate $\tilde{\mathbf{g}}(\mathbf{Y}_a)$ of the pdf $\mathbf{g}(\mathbf{Y}_a)$ from the set of test results $\mathbf{y}_a^r,\ r = 1, \ldots, M$.
5. Compare the estimates $\tilde{\mathbf{g}}(\mathbf{Y})$ and $\tilde{\mathbf{g}}(\mathbf{Y}_a)$ (Section 2.4).

The application of the approach to the Gaussian peak fitting problem (Section 3.3) is given in Section 6.

6 Case Study: Gaussian Peak Fitting

In this section we use the example problem of Gaussian peak fitting (Section 3.3) to illustrate those methods for algorithm testing described in Section 5. We suppose the Gaussian peak fitting problem is addressed by seeking the solutions to two problems specified by different computational aims (here, C_1 and C_2), and we wish to investigate the consequences of using a solution to the (simpler) problem specified by C_2 as a substitute for a solution to that specified by C_1.

In particular, we cover the following aspects:

- specifications of the computational aims,
- parametrisation of the Gaussian peak model, with its influence on algorithms for solving the problems specified by the computational aims,

- software implementations of algorithms for solving the problems,
- data generators for the computational aims,
- software testing, encompassing both numerical software testing and algorithm testing, using the software implementations and data generators, and
- algorithm testing using simulation.

6.1 Computational aims

We consider the following computational aims for the Gaussian peak fitting problem (see Section 3.3).

6.1.1 Computational aim C_1

Given measurement data $(x_i, y_i),\ i = 1, \ldots, m$, find values for the parameters A, $\bar{x}$ and s that minimize

$$\sum_{i=1}^{m} \{y_i - y(x_i)\}^2 ,$$

where

$$y(x) = A \exp\left(-\frac{(x-\bar{x})^2}{2s^2}\right).$$

6.1.2 Computational aim C_2

Given measurement data $(x_i, y_i),\ i = 1, \ldots, m$, find values for the parameters A, $\bar{x}$ and s that minimize

$$\sum_{i \in I} \{\ln y_i - \ln y(x_i)\}^2 ,$$

where $I = \{i : y_i > 0\}$.

6.2 Parametrisation

In terms of the parameters A (amplitude), $\bar{x}$ (location) and s (width), a Gaussian peak model is defined (see above) by

$$y(x) = A \exp\left(-\frac{(x-\bar{x})^2}{2s^2}\right).$$

Recalling (10), a Gaussian peak model may also be written in the form

$$y(x) = \exp\left(a_0 + a_1 x + a_2 x^2\right), \quad a_2 < 0.$$

The values for the parameters A, $\bar{x}$ and s in the natural parametrisation of the Gaussian peak model are then recovered from the values of a_0, a_1 and a_2 using the following relationships:

$$A = \exp\left(a_0 - \frac{a_1^2}{4a_2}\right), \quad \bar{x} = -\frac{a_1}{2a_2}, \quad s = \sqrt{-\frac{1}{2a_2}}.$$

For reasons of numerical stability, a better representation is the *centred* form

$$y(x) = \exp\left(b_0 + b_1(x - x_0) + b_2(x - x_0)^2\right), \quad b_2 < 0, \tag{19}$$

where

$$A = \exp\left(b_0 - \frac{b_1^2}{4b_2}\right), \quad \bar{x} = x_0 - \frac{b_1}{2b_2}, \quad s = \sqrt{-\frac{1}{2b_2}},$$

and x_0 is set equal to the arithmetic mean of the data abscissa values, $x_i,\ i = 1, \ldots, m$.

For the purposes of the software implementations described below, the Gaussian peak model is described in terms of parameters $\mathbf{a} = (a_0, a_1, a_2)^{\mathrm{T}}$ or $\mathbf{b} = (b_0, b_1, b_2)^{\mathrm{T}}$, with values for A, $\bar{x}$ and s obtained using the expressions given above. Additionally, the residual deviations $y_i - y(x_i),\ i = 1, \ldots, m$, i.e., the differences between the measurement data values and model values, provide a representation of the model that is *independent* of any particular parametrisation of $y(x)$.

6.3 Software implementations of the computational aims

6.3.1 Software S_1

In terms of the parametrisation (19), the computational aim C_1 reduces to the nonlinear least-squares problem

$$\min_{\mathbf{b}} \sum_{i=1}^{m} \left\{ y_i - \exp\left(b_0 + b_1(x_i - x_0) + b_2(x_i - x_0)^2\right) \right\}^2.$$

Software S_1 solves this problem using a Gauss-Newton algorithm [15]. The Gauss-Newton algorithm is an iterative algorithm where at each iteration the linear least-squares problem

$$J\,\delta\mathbf{b} = -\mathbf{r}$$

is solved, where $\mathbf{r}$ contains the residual deviations

$$r_i = y_i - y(x_i), \quad i = 1, \ldots, m,$$

and J is the (Jacobian) matrix of partial derivatives

$$\frac{\partial r_i}{\partial b_j}, \quad i = 1, \ldots, m \quad j = 1, 2, 3,$$

both evaluated for the current estimates of the parameters $\mathbf{b}$. The vector $\delta\mathbf{b}$ contains the Gauss-Newton step which is used to update the current estimates of the parameters $\mathbf{b}$ to provide new estimates. In the software implementation S_1, initial estimates of the parameters are provided by the software S_2 (see below) for solving the problem specified by the computational aim C_2, and the iteration continues until there is no further decrease in the residual sum of squares

$$\sum_{i=1}^{m} r_i^2.$$

6.3.2 Software S_2

In terms of the parametrisation (19), the computational aim C_2 reduces to the linear least-squares problem

$$\min_{\mathbf{b}} \sum_{i \in I} \left\{ \ln y_i - \left(b_0 + b_1(x_i - x_0) + b_2(x_i - x_0)^2 \right) \right\}^2.$$

This is equivalent to finding the least-squares solution to the overdetermined system of linear equations

$$K\mathbf{b} = \mathbf{c},$$

where $\mathbf{c}$ contains the elements $\ln y_i,\ i \in I$ and K is the matrix of partial derivatives

$$\frac{\partial q_i}{\partial b_j}, \quad q_i = \ln y_i - \ln y(x_i), \quad i \in I, \quad j = 1, 2, 3.$$

Software S_2 solves this problem using an orthogonal matrix factorisation of K [12, section 4] (Section 3.2).

6.4 Data generators for the computational aims

6.4.1 Data generator G_1

A solution to the problem specified by the computational aim C_1 is characterised by the condition

$$J^{\mathrm{T}}\mathbf{r} = \mathbf{0},$$

where both $\mathbf{r}$ and J are evaluated at the solution. Based on this condition, a procedure for generating reference data $y_i,\ i = 1, \ldots, m$, for which the solution

to the problem specified by the computational aim C_1 is given *a priori* takes the following form [5, 13]:

Given values for A, $\bar{x}$, s, $x_i : \ i = 1, \ldots, m$:

$G_1\mathbf{a}$ Evaluate the model values $y(x_i), \ i = 1, \ldots, m$.

$G_1\mathbf{b}$ Evaluate the Jacobian matrix J.

$G_1\mathbf{c}$ Form a set of target residual deviations $e_{0,i}, \ i = 1, \ldots, m$.

$G_1\mathbf{d}$ Form the null space N_1 for J^{T}.

$G_1\mathbf{e}$ Form the residual vector $\mathbf{e}_1 = N_1\mathbf{u}$, where $\mathbf{u} = N_1^{\mathrm{T}}\mathbf{e}_0$.

$G_1\mathbf{f}$ Form the measurement values $y_{1,i} = y(x_i) + e_{1,i}, \ i = 1, \ldots, m$.

The resulting data set will have a known solution (defined by the values A, $\bar{x}$ and s), and characterised by a known vector $\mathbf{e}_1$ of residual deviations, that is "close" (in a least-squares sense) to the vector $\mathbf{e}_0$ of target residual deviations. For example, the target residuals may be chosen to be random samples from a Gaussian (normal) distribution with mean zero and variance σ^2, and thus to mimic measurement errors that are likely to be encountered in practice.

6.4.2 Data generator G_2

A solution to the problem specified by the computational aim C_2 can be similarly characterised, viz., by the condition

$$K^{\mathrm{T}}\mathbf{q} = \mathbf{0},$$

where both $\mathbf{q}$ and K are evaluated at the solution. Based on this condition, a procedure for generating reference data $y_i, \ i = 1, \ldots, m$, for which the solution to the problem specified by the computational aim C_2 is given *a priori* takes the following form:

Given values for A, $\bar{x}$, s, $x_i : \ i = 1, \ldots, m$:

$G_2\mathbf{a}$ Evaluate the model values $y(x_i), \ i = 1, \ldots, m$.

$G_2\mathbf{b}$ Evaluate the Jacobian matrix K.

$G_2\mathbf{c}$ Form a set of target residual deviations $e_{0,i}, \ i = 1, \ldots, m$.

$G_2\mathbf{d}$ Form the null space N_2 for K^{T}.

$G_2\mathbf{e}$ Form the vector $\mathbf{q} = N_2\mathbf{u}$, where $\mathbf{u} = N_2^{\mathrm{T}}\mathbf{v}$ and

$$v_i = \ln\left(y(x_i) + e_{0,i}\right) - \ln y(x_i) = \ln \frac{y(x_i) + e_{0,i}}{y(x_i)}, \quad i = 1, \ldots, m.$$

$G_2\mathbf{f}$ Form the measurement values $y_{2,i} = \exp\left(\ln y(x_i) + q_i\right),\ i = 1, \ldots, m.$

$G_2\mathbf{g}$ Form the residual vector $\mathbf{e}_2$ with elements $e_{2,i} = y_i - y(x_i),\ i = 1, \ldots, m.$

The resulting data set will have a known solution (defined by the values A, $\bar{x}$ and s). In addition, from Step $G_2\mathbf{f}$,

$$y_{2,i} = y(x_i) \exp q_i,$$

and so

$$y_{2,i} \approx y(x_i) \exp\left(\ln \frac{y(x_i) + e_{0,i}}{y(x_i)}\right)$$

since, from Step $G_2\mathbf{e}$,

$$q_i \approx v_i = \ln \frac{y(x_i) + e_{0,i}}{y(x_i)}.$$

Consequently,

$$y_{2,i} \approx y(x_i) + e_{0,i},$$

and so the reference data is characterised by a known vector $\mathbf{e}_2$ of residual deviations that are "close" to the target residual deviations $\mathbf{e}_0$. As before, the target residual deviations may be chosen to mimic measurement errors that are likely to be encountered in practice. Alternatively, by setting $\mathbf{e}_0$ in G_2 equal to the residual deviations $\mathbf{e}_1$ returned by G_1, the reference data for the problem specified by C_2 will be close to that for the problem specified by C_1 (Section 5.3).

6.5 Analysis of computational aims

The problem specified by C_1 is to solve

$$\min_{A,\bar{x},s} \sum_{i=1}^{m} r_i^2, \quad r_i = y_i - y(x_i), \quad i = 1, \ldots, m,$$

and that specified by C_2 is to solve

$$\min_{A,\bar{x},s} \sum_{i \in I} q_i^2, \quad q_i = \ln y_i - \ln y(x_i), \quad i \in I.$$

We seek the conditions, i.e., the requirements on the measurement data $(x_i, y_i),\ i = 1, \ldots, m$, for which the two problems give (a) identical solutions, and (b) comparable solutions.[9]

[9] Note that even if the problems deliver comparable estimates of the values of the parameters A, $\bar{x}$ and s, the uncertainties associated with the estimates may be quite different. We do not consider here the conditions under which the problems deliver estimates *and* associated uncertainties that are comparable.

Clearly, if the data is such that

$$y_i = y(x_i), \quad i = 1, \ldots, m,$$

for certain values of A, $\bar{x}$ and s, then the problems will deliver identical solutions, for in this case

$$r_i = q_i = 0, \quad i = 1, \ldots, m,$$

at the solution.

Now,

$$\ln y_i = \ln \left(y(x_i) + r_i \right),$$

and so by a first order Taylor series expansion,

$$\ln y_i \approx \ln y(x_i) + \frac{1}{y(x_i)} r_i.$$

Therefore,

$$q_i \approx \frac{1}{y(x_i)} r_i. \tag{20}$$

Only if the values $x_i,\ i = 1, \ldots, m$, are such that $y(x_i)$ is essentially constant can we expect the problems to be approximately equivalent. This is the case, for example, if the measurements are restricted to a tail of the underlying Gaussian peak function.[10]

Note that the problem

$$\min_{A,\bar{x},s} \ \sum_{i \in I} y_i^2 q_i^2,$$

which is a *weighted* linear least-squares problem, with weights y_i used in place of the (unknown) values $y(x_i)$ in (20), can be expected to perform better than the unweighted problem as an approximation to the problem specified by C_1, with its performance improving as the residual deviations $r_i,\ i = 1, \ldots, m$, tend to zero, i.e., for smaller amounts σ of measurement noise.

6.6 Software Testing

For a given value of σ, and setting

$$A = 1,\ \bar{x} = 1000,\ s = 1,\ m = 101,$$

with $x_i,\ i = 1, \ldots, m$, linearly spaced in the interval $[\bar{x} - 3s, \bar{x} + 3s]$, the data generator G_1 is used to generate reference data $y_{1,i},\ i = 1, \ldots, m$, for the problem

[10] However, in such circumstances the data will not define the underlying Gaussian peak function very well, and the problem of determining the parameters of this function can be expected to be ill-conditioned.

specified by C_1, with the target residuals $\mathbf{e}_0$ in G_1 chosen to be random samples from a Gaussian (normal) distribution with mean zero and variance σ^2. Similarly, the data generator G_2 is used to generate reference data $y_{2,i},\ i = 1, \ldots, m$, for the problem specified by C_2, with the target residuals $\mathbf{e}_0$ in G_2 set equal to the residual deviations returned by G_1.

The following tests are then undertaken:

1. Apply software S_1 to the reference data $(x_i, y_{1,i}),\ i = 1, \ldots, m$. The solution returned by S_1 is represented by the vector $\mathbf{r}_1$ of residual deviations, and is compared with the reference solution $\mathbf{e}_1$ using the measures

$$d_1 = \frac{\|\mathbf{r}_1 - \mathbf{e}_1\|}{\sqrt{m}},$$

and

$$P_1 = \log_{10}\left(1 + \frac{d_1\sqrt{m}}{\|\mathbf{y}\|\eta}\right).$$

Here, d_1 measures the root-mean-square error between $\mathbf{r}_1$ and $\mathbf{e}_1$, and P_1 is a performance measure that indicates the number of significant figures of accuracy lost by the software compared with a reference implementation of software to solve the problem specified by C_1 (Section 2.1).

Software S_1 is applied with the model parameter x_0 set equal to the arithmetic mean of the data abscissa values

2. Apply software S_2 to the reference data $(x_i, y_{2,i}),\ i = 1, \ldots, m$. The solution returned by S_2 is represented by the vector $\mathbf{r}_2$ of residual deviations, and is compared with the reference solution $\mathbf{e}_2$ using the measures

$$d_2 = \frac{\|\mathbf{r}_2 - \mathbf{e}_2\|}{\sqrt{m}},$$

and

$$P_2 = \log_{10}\left(1 + \frac{d_2\sqrt{m}}{\|\mathbf{y}\|\eta}\right).$$

Software S_2 is applied with the model parameter x_0 set equal to (a) the arithmetic mean of the data abscissa values, and (b) zero.

3. Apply software S_2 to the reference data $(x_i, y_{1,i}),\ i = 1, \ldots, m$. The solution returned by S_2 is represented by the vector $\mathbf{r}_3$ of residual deviations, and is compared with the reference solution $\mathbf{e}_1$ using the measures

$$d_3 = \frac{\|\mathbf{r}_3 - \mathbf{e}_1\|}{\sqrt{m}},$$

and

$$P_3 = \log_{10}\left(1 + \frac{d_3\sqrt{m}}{\|\mathbf{y}\|\eta}\right).$$

Software S_2 is applied with the model parameter x_0 set equal to (a) the arithmetic mean of the data abscissa values, and (b) zero.

4. Calculate the distance between the reference data sets for the problems specified by the computational aims C_1 and C_2 using the measure

$$d_4 = \frac{\|\mathbf{y}_1 - \mathbf{y}_2\|}{\sqrt{m}}.$$

Here, d_4 measures the root-mean-square error between the reference data sets $\mathbf{y}_1$ and $\mathbf{y}_2$.

The above data generation procedure and tests are repeated a number of times for each of a number of values of σ in the range $[0, 0.01]$.

Figure 5 shows how the values of the measures d_1 and P_1, corresponding to the software S_1, vary with σ. The root-mean-square error between the residual deviations returned by the software S_1 and the reference values for the residual deviations is approximately 10^{-14}. The performance measure takes values of approximately two, indicating that the software S_1 is losing roughly two significant figures of accuracy compared with reference, i.e., optimally stable, software for solving the problem specified by C_1. There is a *slight* negative correlation between the values of d_1 (P_1) and σ. The results indicate that for the reference data sets considered here the software S_1 is close to a reference implementation of software for solving the problem specified by C_1.

Figures 6 and 7 show how the values of the measures d_2 and P_2, corresponding to the software S_2, vary with σ for the different ways in which the model parameter x_0 is set. The results indicate that for the reference data sets considered here the software S_2 with x_0 equal to the arithmetic mean of the data abscissa values performs almost as well as a reference implementation of software for solving the problem specified by C_2. On the other hand, the software S_2 with x_0 equal to zero is losing between six and seven significant figures of accuracy compared with reference software for solving this problem. The software S_2 with x_0 set equal to zero cannot be regarded as a reference implementation. In both cases there is no strong dependence of the values of d_2 and P_2 on σ.

The results shown in Figures 5, 6 and 7, and discussed above, concern *numerical software testing* of the software S_1 and S_2 against their respective computational aims. In contrast, the results shown in Figures 8 and 9 are based on comparing the results returned by the software S_2 with reference results for the problem specified by C_1. The results indicate that the software S_2, irrespective of the way the model

parameter x_0 is set, does not provide accurate solutions to the problem specified by C_1. Because the software S_2 solves the problem specified by C_2 to an accuracy that is appreciably greater than that for the problem specified by C_1, we conclude that the results shown in these figures are predominantly a consequence of the difference between the computational aims rather than being attributable to the software implementations. This conclusion applies irrespective of how the model parameter x_0 is set, and we are therefore able, in this example, to deduce useful information about the computational aims from a poor algorithm for solving the problem specified by C_2. The results shown in Figures 8 and 9 concern *algorithm testing*.

Figure 10 shows how the values of the measure d_4 vary with σ. The value of d_4 measures the (root-mean-square) "distance" between two data sets having the same solution, for one data set the solution is for the problem specified by computational aim C_1, and for the other for that specified by C_2. The measure provides information about the departure between the computational aims independently of software for solving the problems specified by the computational aims. The results are very similar to those shown in Figures 8 and 9, confirming that these results are measuring the departure between the computational aims. There is a strong dependence of the values of d_4 on σ. Furthermore, the values of the measure d_4 are of a similar magnitude to σ. This indicates that there exist data sets that can differ by an amount as large as σ and that for the problems specified by C_1 and C_2, respectively, have identical solutions.

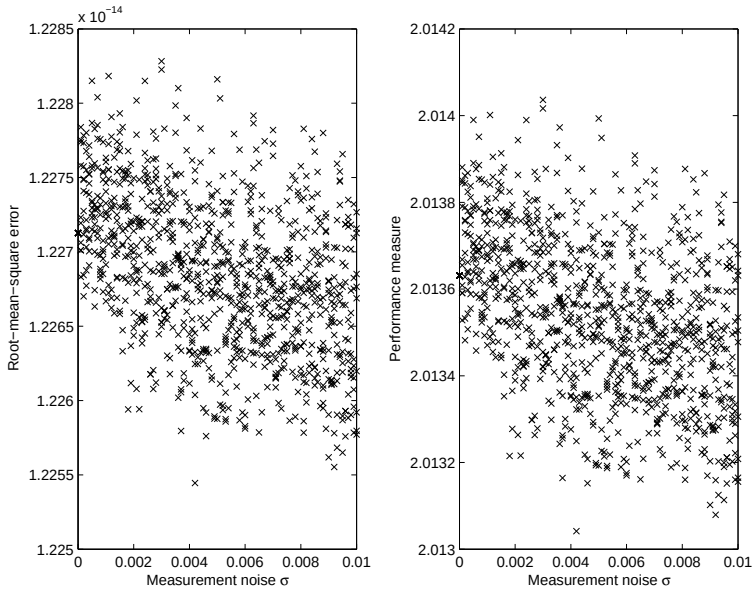

Figure 5: Testing the numerical correctness of software S_1 to solve the problem specified by the computational aim C_1.

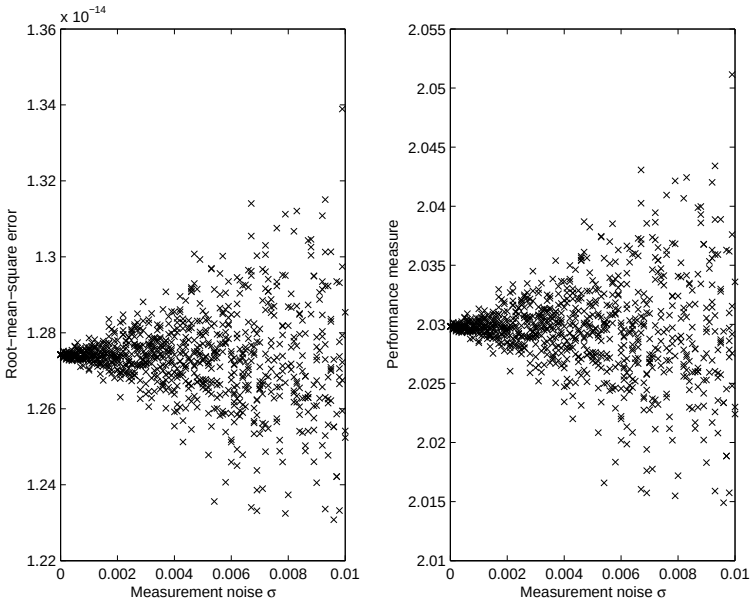

Figure 6: Testing the numerical correctness of software S_2 to solve the problem specified by the computational aim C_2. The model parameter x_0 is set equal to the arithmetic mean of the data abscissa values $x_i,\ i = 1, \ldots, m$.

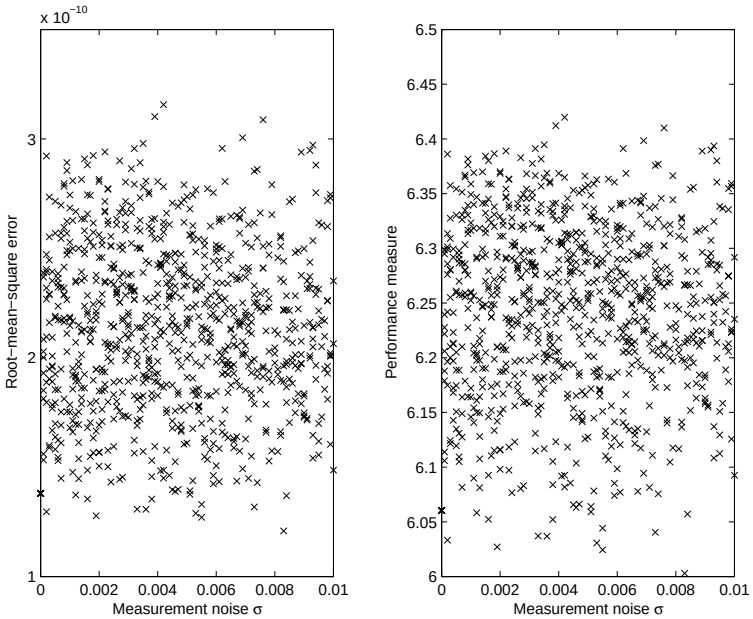

Figure 7: Testing the numerical correctness of software S_2 to solve the problem specified by the computational aim C_2. The model parameter x_0 is set equal to zero.

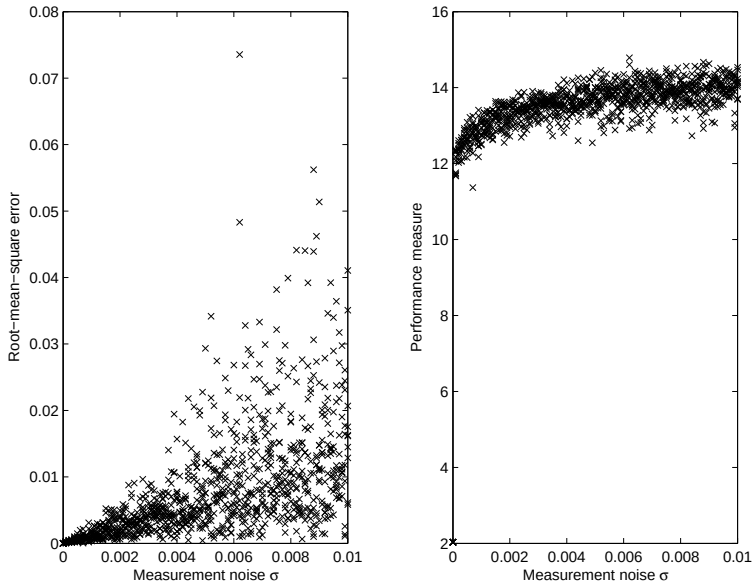

Figure 8: Performance of software S_2 on reference data for the problem specified by the computational aim C_1. The model parameter x_0 is set equal to the arithmetic mean of the data abscissa values $x_i,\ i = 1, \ldots, m$.

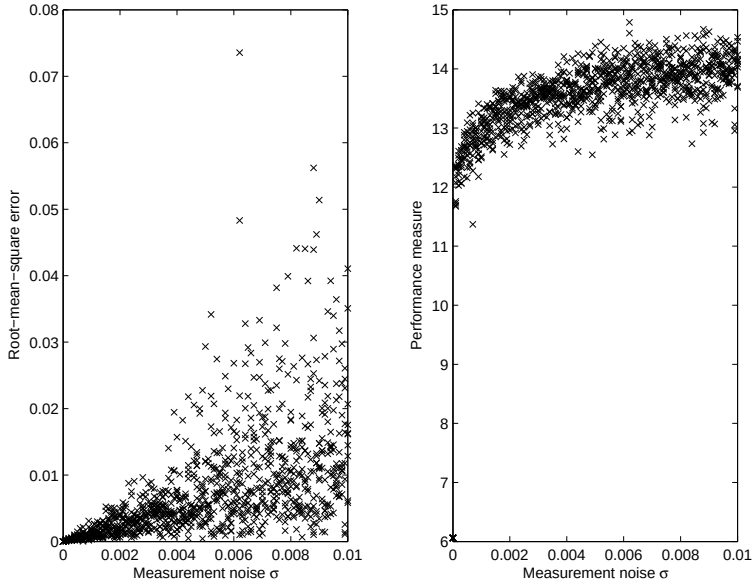

Figure 9: Performance of software S_2 on reference data for the problem specified by the computational aim C_1. The model parameter x_0 is set equal to zero.

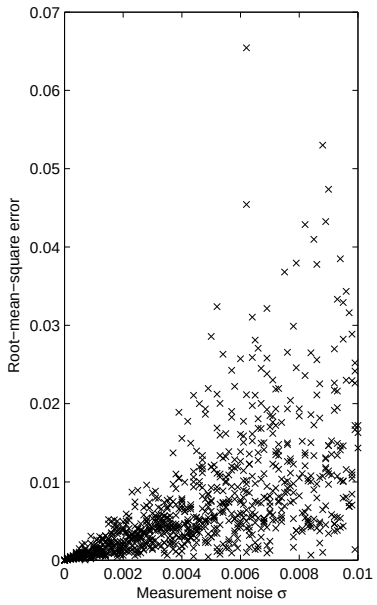

Figure 10: Comparing computational aims for the Gaussian peak fitting problem using data generation software.

6.7 Simulation

Given values for A, $\bar{x}$, s, $x_i,\ i = 1, \ldots, m$, σ and M, repeat the following steps for $r = 1, \ldots, M$:

1. Construct the data set $(x_i, y_i^r),\ i = 1, \ldots, m$, by evaluating

$$y_i^r = A \exp\left(-\frac{(x_i - \bar{x})^2}{2s^2}\right) + e_i^r, \quad i = 1, \ldots, m,$$

   where e_i^r is a random sample from $N(0, \sigma^2)$, a Gaussian (normal) distribution with mean zero and variance σ^2.

2. Apply software S_1 (with the model parameter x_0 set equal to the arithmetic mean of the data abscissa values) to the data set to obtain results $(A_1^r, \bar{x}_1^r, s_1^r)$.

3. Apply software S_2 (with the model parameter x_0 set equal to the arithmetic mean of the data abscissa values) to the data set to obtain results $(A_2^r, \bar{x}_2^r, s_2^r)$.

Figure 11 shows, in the form of histograms derived from the results

$$(A_1^r, \bar{x}_1^r, s_1^r),\ r = 1, \ldots, M,$$

estimates of the probability density functions (pdf's) for the values of A_1, $\bar{x}_1$ and s_1, the solution to the problem specified by the computational aim C_1 returned by the software S_1.

Similarly, Figure 12 shows estimates of the pdf's for the values of A_2, $\bar{x}_2$ and s_2, the solution to the problem specified by the computational aim C_2 returned by the software S_2. The results shown in the two figures correspond to setting

$$A = 1,\ \bar{x} = 1000,\ s = 1,\ m = 101,\ \sigma = 0.01,\ M = 50,000,$$

with $x_i,\ i = 1, \ldots, m$, linearly spaced in the interval $[\bar{x} - 3s, \bar{x} + 3s]$.

The pdf's shown in the figures describe the inexactness of the values of the measurement results, viz., the values of A, $\bar{x}$ and s, derived from the same information about the measurement data but solving problems specified by different computational aims to obtain the measurement results from that data. Where the measurement results are obtained by solving the problem specified by C_2, it is evident that the spread of possible values of the measurement results is considerably greater than that obtained by solving the problem specified by C_1. There is a high probability of obtaining a solution to the problem specified by C_2 that has a low probability of arising as a solution to the problem specified by C_1. In addition, the pdf's for the values of A_2 and s_2 illustrate an asymmetry which makes the description of the uncertainties associated with the values of these parameters less straightforward than for the other parameters.

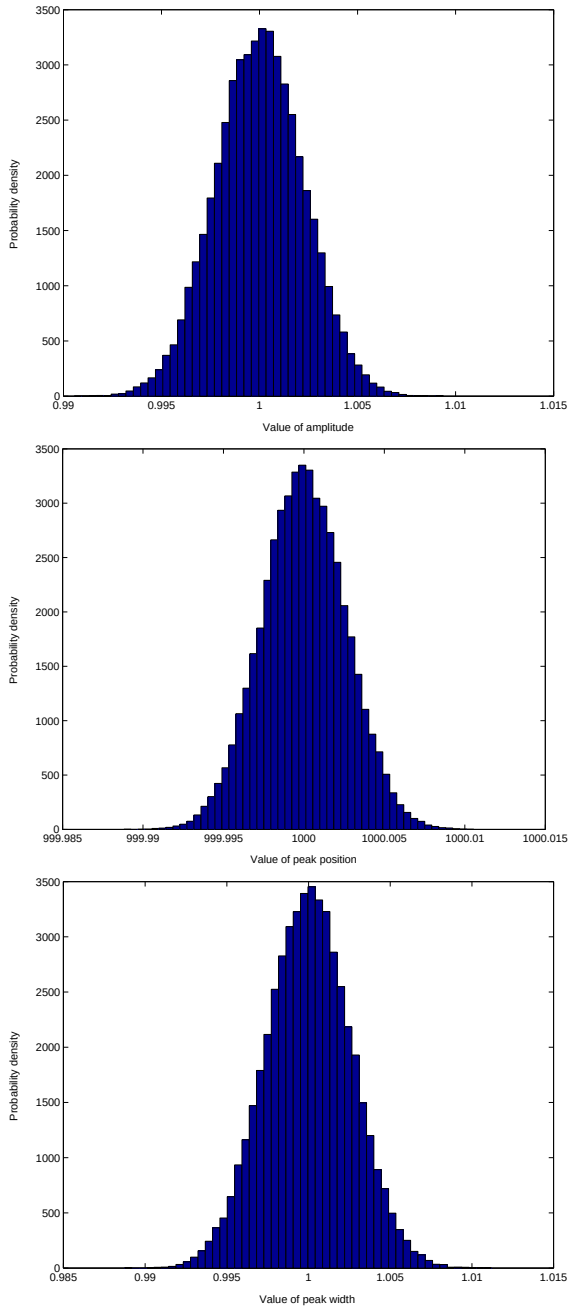

Figure 11: Estimates of the pdf's for the values of A_1, $\bar{x}_1$ and s_1, the solution to the computational aim C_1 returned by the software S_1.

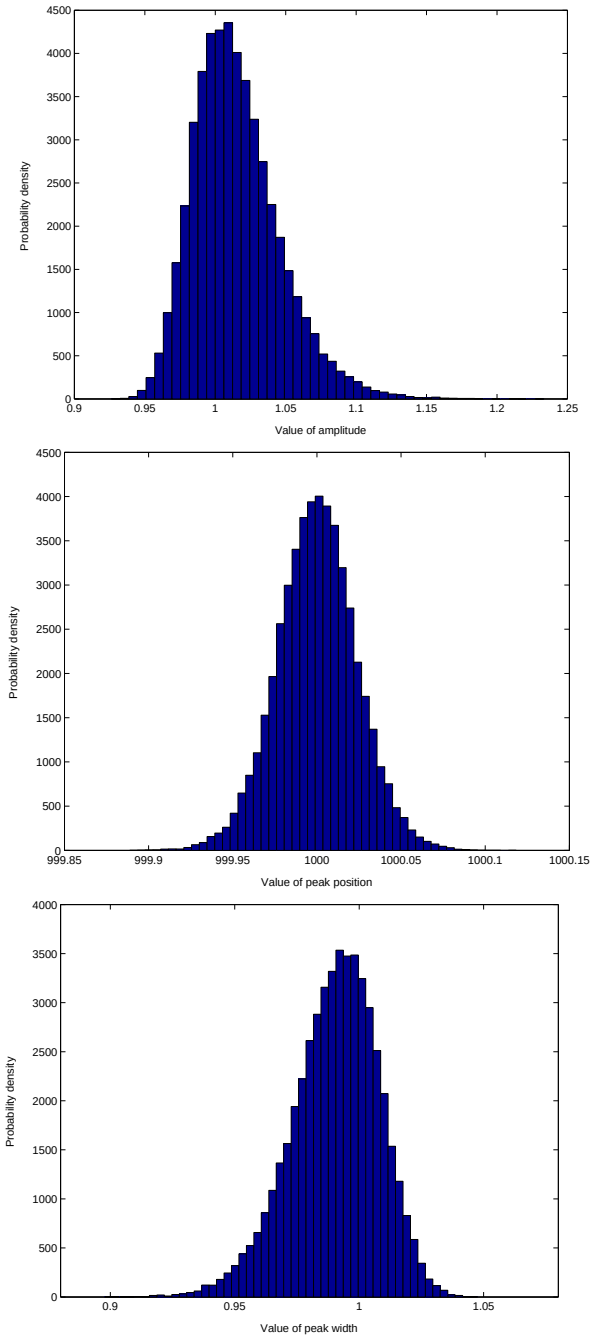

Figure 12: Estimates of the pdf's for the values of A_2, $\bar{x}_2$ and s_2, the solution to the computational aim C_2 returned by the software S_2.

7 Conclusions

This report has presented a framework for describing the activities of algorithm testing and numerical software testing, and how these activities contribute to the validation of the software implementation of an algorithm for achieving a given computational aim. The framework emphasises the use of mathematical analysis to compare algorithms and numerical software testing to compare implementations. A consequence is the importance of understanding, in any particular instance, the nature of the object (algorithm or implementation) to be validated, as well as where errors are likely to have been introduced.

It is apparent that there is much scope in the area of algorithm testing for using and extending the methods developed for numerical software testing. However, it is also clear that it is costly and labour intensive to develop new data sets and to undertake simulations to validate new algorithms. Furthermore, there is no real scope for automating the mathematical analysis of computational aims necessary for comparing algorithms and their computational aims. Mathematical analysis is also generally very labour intensive. The best way to avoid the need to compare algorithms and their computational aims is to use established algorithms which are supported by the necessary mathematical analysis. The use of new algorithms has considerable hidden costs, particularly if they are to be used in a high integrity traceable environment such as metrology.

A testing service to cover algorithms for submission to Standards and software libraries would involve both mathematical analysis and numerical software testing. Both activities are labour intensive – mathematical analysis particularly so. A service that concentrated on software based on existing algorithms, and where the use of those algorithms was properly documented, is the one most likely to be commercially viable.

8 Acknowledgements

The work described here was supported by the National Measurement System Policy Unit of the UK Department of Trade and Industry as part of its NMS Software Support for Metrology programme.

We thank Sven Hammarling of NAG Ltd for helpful discussions about this work.

References

[1] ISO 4287. *Geometrical Product Specifications (GPS) – Surface texture: Profile method – Terms, definitions and surface texture parameters*. International Organisation for Standardisation, Geneva, Switzerland, 1997.

[2] ISO 5725. *Accuracy (trueness and precision) of measurement methods and results*. International Organisation for Standardisation, Geneva, Switzerland, 1994.

[3] R. M. Barker. Best Practice Guide No. 5. Software Re-use: Guide to METROS. Technical report, National Physical Laboratory, Teddington, UK, 2000.

[4] BIPM, IEC, IFCC, ISO, IUPAC, IUPAP, and OIML. Guide to the Expression of Uncertainty in Measurement, 1995. ISBN 92-67-10188-9, Second Edition.

[5] H. R. Cook, M. G. Cox, M. P. Dainton, and P. M. Harris. A methodology for testing spreadsheets and other packages used in metrology. Technical Report CMSC 25/99, National Physical Laboratory, Teddington, UK, 1999.

[6] H. R. Cook, M. G. Cox, M. P. Dainton, and P. M. Harris. Testing spreadsheets and other packages used in metrology: A case study. Technical Report CMSC 26/99, National Physical Laboratory, Teddington, UK, 1999.

[7] M. G. Cox, M. P. Dainton, A. B. Forbes, P. M. Harris, P. M. Schwenke, B. R. L Siebert, and W. Wöger. Use of Monte Carlo Simulation for uncertainty evaluation in metrology. In P. Ciarlini, M. G. Cox, E. Filipe, F. Pavese, and D. Richter, editors, *Advanced Mathematical Tools in Metrology V. Series on Advances in Mathematics for Applied Sciences Vol. 57*, pages 93–104, Singapore, 2001. World Scientific.

[8] M. G. Cox, M. P. Dainton, and P. M. Harris. Testing functions for linear regression. Technical Report CMSC 08/00, National Physical Laboratory, Teddington, UK, 2000.

[9] M. G. Cox, M. P. Dainton, and P. M. Harris. Testing functions for the calculation of standard deviation. Technical Report CMSC 07/00, National Physical Laboratory, Teddington, UK, 2000.

[10] M. G. Cox, M. P. Dainton, and P. M. Harris. Software specifications for uncertainty evaluation and associated statistical analysis. Technical Report CMSC 10/01, National Physical Laboratory, Teddington, UK, 2001.

[11] M. G. Cox, M. P. Dainton, and P. M. Harris. Software Support for Metrology Best Practice Guide No. 6. Uncertainty and Statistical Modelling. Technical report, National Physical Laboratory, Teddington, 2001.

[12] M. G. Cox, A. B. Forbes, and P. M. Harris. Software Support for Metrolgy Best Practice Guide No. 4. Discrete Modelling. Technical report, National Physical Laboratory, Teddington, 2000.

[13] M. G. Cox and P. M. Harris. Design and use of reference data sets for testing scientific software. *Analytica Chimica Acta*, 380:339 – 351, 1999.

[14] A. B. Forbes and P. M. Harris. Estimation algorithms in the calculation of effective area of pressure balances. *Metrologia*, 36:689–692, 1999.

[15] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, London, 1981.

[16] J. C. Legras and P. Delajoud. Comparaison des étalons de pression du Gosstandart (URSS) et du Bureau National de Métrologie (France) avec la balance de transfert de pression du Bureau National de Métrologie (Gramme 1 à 7 MPa). *Bulletin BNM*, pages 6–11, 1977.

[17] S. L. Lewis and G. N. Peggs. *The Pressure Balance: A Practical Guide to its Use*. HMSO, London, second edition, 1992.

[18] D. I. Simpson. Computerized techniques for calibrating pressure balances. *Metrologia*, 30:655–658, 1993.

[19] R. Sym. An assessment of the extent to which metrology algorithms are used in standands. signalsfromnoise.com, 2002. http://www.npl.co.uk/ssfm/download/documents/roger_sym_report.pdf.

[20] D. J. Whitehouse. *Handbook of Surface Metrology*. IOP Publishing Ltd., 1993.

[21] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice-Hall, Engelwood Cliffs, 1963.