NPL Report CISE 25/99

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

A METHODOLOGY FOR TESTING SPREADSHEETS AND OTHER PACKAGES USED IN METROLOGY

BY

H R COOK, M G COX, M P DAINTON and P M HARRIS

September 1999

A Methodology for Testing Spreadsheets and Other Packages Used in Metrology

H R Cook, M G Cox, M P Dainton and P M Harris
Centre for Information Systems Engineering

September 1999

ABSTRACT

A general methodology for testing the numerical accuracy of scientific software is presented. The basis of the approach is the design and use of reference data sets and corresponding reference results to undertake black-box testing. Reference data sets and results are generated in a manner consistent with the functional specification of the problem addressed by the test software. In order to make the testing representative of a particular metrology application, and to consider "fitness for purpose" of the test software, data sets are produced with application-specific properties and various "degrees of difficulty". The results returned by the software for the reference data are compared objectively with the reference results. Quality metrics are used for this purpose that account for the key aspects of the problem.

Complementary tests that do not require the availability of reference data are also described. These include consistency and continuity checks, spot checks against tabulated values, and checks of solution characterisations.

This report constitutes part of the deliverable of Project 2.1 "Testing Spreadsheets and Other Packages Used in Metrology" within the UK Department of Industry's National Measurement System *Software Support for Metrology* Programme 1998–2001.

ISSN 1361-407X

Authorised by Dr Dave Rayner,
Head of the Centre for Information Systems Engineering

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

CONTENTS

1. Introduction

Although scientific software is very widely used throughout metrology, it is rarely tested in an objective and impartial manner. There is a growing need, driven to some extent by Quality Management Systems, to demonstrate that software used by scientists is fit for purpose and especially that the results it produces are correct, to within a prescribed accuracy, for the problems purportedly solved. Some approaches to testing the numerical accuracy of scientific software have been developed. These address the individual software modules called directly by scientists or as part of software packages used in science [1, 2, 3]. Even when the state of the art has developed to the extent that integrated systems can be tested properly, it will remain necessary to test individual modules, because of their importance in themselves, and their considerable use by software packages. Integrated-systems testing concentrates on the interfaces between modules and such careful testing in conjunction with individual module tests constitutes a key part of an overall test.

Software used in scientific disciplines can be unreliable because it implements numerical algorithms that are unstable or not robust. Some of the reasons [4] for such failings are

1. failure to scale, translate, normalise or otherwise transform the input data appropriately before solution, and to perform the inverse operation if necessary following solution,
2. the use of an unstable parametrisation of the problem,
3. the use of a solution process that exacerbates the inherent (natural) ill-conditioning of the problem, and
4. a poor choice of formula from a set of mathematically (but not numerically) equivalent forms.

Some remarks about problem parametrisation (that relate to 1 and 2 above) are made in Section 1.1, and about problem conditioning (that relate to 3 and 4 above) in Section 1.2.

The purpose of this report is to describe a general methodology for testing the numerical accuracy of scientific software. The basis of the approach is the design and use of reference data sets and corresponding reference results to undertake black-box testing. The approach enables reference data sets and results to be generated in a manner consistent with the functional specification of the problem addressed by the software. In addition, data sets corresponding to problems with various “degrees of difficulty”, or with application-specific properties, may be produced. The results returned by the software for the reference data are compared objectively with the reference results. Quality metrics are used for this purpose that account for the key aspects of the problem. The companion reports [6, 7] describe the application of the methodology, respectively, in a case study concerned with testing software for the problem “Fit a Gaussian peak to measurement data”, and to testing the in-built functions of the Excel spreadsheet package.

The report is organised as follows. In Section 2 we discuss the requirements for testing placed on the user or developer of the test software: the availability of an (unambiguous) specification for the software and an implementation of that specification that permits the testing methodology to be applied. Sections 3 and 4 cover, respectively, the design and use of reference data sets and corresponding results to undertake black-box testing. The emphasis here is on showing that the testing approach is objective, robust and defensible. There are other, complementary tests of numerical accuracy that are also valuable, and some of these are listed in Section 5. Some examples of the application of the described methodology are given in Section 6. Our conclusions are given in Section 7.

1.1 Problem Parametrisation

Problem parametrisation can readily be demonstrated using the example of a straight line. A straight line can be expressed in a variety of ways, including

$$y = a_1 + a_2 x, \tag{1}$$

where a_1 is the y-axis intercept and a_2 is the gradient, and

$$y = a_1 + a_2 (x - c), \tag{2}$$

where a_2 is the gradient as before, but a_1 now represents the intercept with the line $x = c$, the value of c being at choice, rather than with $x = 0$, the y-axis. Parametrisation (2) is clearly a generalisation of (1). This generality can be used to advantage in designing an algorithm for straight-line fitting, by selecting a value for c that yields desirable properties. The choice of c as the arithmetic mean of the data abscissa x_i has several such properties [1, p120]. There is an alternative to re-parametrisation in this case, viz., data shifting. If the data abscissa values x_i are replaced before fitting by $x_i - c$, the form (1) then obtains.

The type of testing considered here can implicitly identify whether it is likely that a sensible parametrisation has or has not been used (or equivalently whether a suitable data transformation has been adopted).

Problem parametrisation can raise a difficulty for the software tester. If there is no widely recognised or "canonical" parametrisation, either the values of the parameters corresponding to the actual parametrisation used by the software must be compared with reference values *based on the same parametrisation*, or some subsidiary or derived quantities must be assessed. The former means that the software tester must be in a position to prepare reference solutions for possibly a wide range of different parametrisations, thus increasing the cost of the test. This is the solution adopted by the International Organisation for Standardisation [5] in its work on preparing a standard for testing software implementations of a class of nonlinear regression algorithms used in industrial inspection. The latter implies that some invariant, e.g., canonical, form must be determined and, moreover, careful analysis carried out to ensure that the test on these quantities implies an acceptable result for the primary parameters. Within the area of regression, for example, the residuals of a correctly computed solution are invariant to the parametrisation and are often suitable for testing purposes. Section 4.1 and [6] considers the use of residuals in testing least-squares regression software.

1.2 Problem Condition

The condition of a problem is a measure that describes the sensitivity of its exact solution to changes in the problem. If small changes in the problem or its data lead to small changes in the solution, the problem is said to be well-conditioned. Otherwise, if small changes in the data lead to large changes in the solution, the problem is said to be ill-conditioned for the data. Conditioning is an inherent characteristic of the mathematical problem, and is independent of any algorithm or software for solving the problem

It is unreasonable to expect even software of the highest quality to deliver results for a problem to the full accuracy indicated by the computational precision η.[1] This would only in general be possible for (some) problems that are perfectly conditioned, i.e., problems for which a small change in the data makes a comparably small change in the results. Problems regularly arise in

[1] 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.

which the ill-conditioning is significant and for which no algorithm, however good, can provide results to the accuracy obtainable for well-conditioned problems.[2]

By using appropriate performance measures (see Section 4.2) to quantify the loss of numerical accuracy caused by the test software over and above that which can be explained by the problem condition, the type of testing considered here can implicitly identify the use of a poor choice of algorithm or formula from a set of mathematically (but not numerically) equivalent forms. We give below two simple examples.

Example. The first example is the calculation of the sample standard deviation *s*. Given the set of data $\{x_i: i = 1, \ldots, m\}$, two mathematically equivalent formulae for evaluating *s* are

$$s = \left\{ \frac{1}{m-1} \sum_{i=1}^{m} (x_i - \bar{x})^2 \right\}^{1/2} \tag{3}$$

and

$$s = \left\{ \frac{1}{m-1} \left(\sum_{i=1}^{m} x_i^2 - m\bar{x}^2 \right) \right\}^{1/2}, \tag{4}$$

where $\bar{x}$ is the sample arithmetic mean defined by

$$\bar{x} = \frac{1}{m} \sum_{i=1}^{m} x_i. \tag{5}$$

The formula (4) has the property that it suffers from subtractive cancellation for data sets with small coefficient of variation $s / \bar{x}$. For instance, if the values of x_i are {0.98, 0.99, 1.00, 1.01, 1.02}, then

$$\bar{x} = 1.00, \quad \sum_{i=1}^{5} x_i^2 = 5.0010, \quad m\bar{x}^2 = 5.0000,$$

and hence three to four significant figures are lost due to subtractive cancellation in forming

$$\sum_{i=1}^{5} x_i^2 - m\bar{x}^2 = 5.0010 - 5.0000 = 0.0010.$$

In some cases, $\sum x_i^2$ will be *equal* to $m\bar{x}^2$ to all figures held on the computer. It is even possible that the computed value of $\sum x_i^2$ will be (marginally) less than that of $m\bar{x}^2$ due to the rounding errors occurred in forming these quantities. In the former case, *s* is computed as zero (even though the x_i are not all identical). In the latter case, some software packages internally replace the computed value of $\sum x_i^2 - m\bar{x}^2$ by zero in order to avoid a failure due to an attempt to take the square root of a negative number. Such cases are *not* pathological. In a comparison [11] of various software packages and electronic calculators, not one of the eight tested on data from a weighing application produced the correct value of *s* to two correct figures, although a perfectly acceptable value would have been obtained by the use of formula (3).

It is shown in [12] and Appendix A.2 that the condition number for the problem of calculating the sample standard deviation is essentially proportional to $\bar{x} / s$. Furthermore, a floating-point error analysis [12, 13] of formulae (3) and (4) shows that the relative error in the value returned by the former behaves like $\bar{x} / s$, whereas for the latter the error behaves like the square of the condition number. Consequently, formula (3) is close to being optimal, i.e., it does not lose

[2] It is emphasised that we assume that calculations are performed using floating-point arithmetic with the stated computational precision η, i.e., not making use of packages which implement higher- or arbitrary-precision arithmetic.

figures unnecessarily, whereas formula (4) can be expected to lose about twice as many figures. Unfortunately, many calculators and software packages implement the unstable formula.

Example. The second example is the calculation of the angle θ between two unit vectors. Given two vectors **a** and **b**, each of unit length, two mathematically equivalent formulae for evaluating θ are

$$\theta = \cos^{-1}(\mathbf{a}^{\mathrm{T}}\mathbf{b}) \tag{6}$$

and

$$\theta = 2\sin^{-1}\left(\frac{1}{2}\|\mathbf{b}-\mathbf{a}\|\right). \tag{7}$$

For vectors that are very nearly parallel, i.e.,

$$\mathbf{a}^{\mathrm{T}}\mathbf{b} = 1-\delta, \quad |\delta| << 1,$$

the form (6) gives, approximately,

$$\cos\theta = 1-\frac{\theta^2}{2} = \mathbf{a}^{\mathrm{T}}\mathbf{b} = 1-\delta,$$

i.e.,

$$\theta = \sqrt{2\delta}.$$

Since the smallest non-zero δ for which 1 − δ is representable is η/2, where η is the computational precision, the smallest θ computable from (6) is $\sqrt{\eta}$, i.e., approximately 10^{-8} radians for IEEE arithmetic. Thus, formula (6) is unable to detect an angle smaller than 10^{-8} radians unless it is computed as zero, whereas the alternative formula (7) can be expected to return accurate values.

The latter example has serious consequences for those concerned with implementing the procedures described in the ISO Draft International Standard [5]. This Standard is concerned with testing software for computing Gaussian best-fit geometric elements to measured data that is used in industrial inspection and geometric tolerancing applications. The Standard requires that the unit direction vector used in the parametrisation of such geometric elements as planes, cylinders and cones and returned by test software is compared with a reference solution by computing the angle between the test and reference vectors. The Standard defines acceptance of the test vector if this angle is smaller than 10^{-8} radians. It is clear from the above analysis that if formula (6) is used to evaluate the angle, this acceptance criterion can never be satisfied no matter how close are the test and reference vectors unless the angle is computed as zero. On the other hand, there is no problem with undertaking the comparison if formula (7) is used.

2. Requirements for Testing

2.1 Specification of the test software

Because detailed documentation on the algorithms employed within software implementations is often unavailable and interaction with the software is limited to "data in, results out'', it is usually necessary to use some form of black-box testing to discern whether any particular item of software is fit for purpose. Such testing can be carried out if there is a specification available of the task carried out by the software. The availability of such a specification is assumed here. The specification may be simple, for example, "*This software calculates the arithmetic mean and standard deviation of a prescribed set of numerical values*". Even in this simple case, the specification is required to be unambiguous. Hence, it is stated that the *arithmetic* mean is of concern (rather than, say, the geometric or the harmonic mean). In a more complicated case,

such as regression, it would be necessary to state such aspects as (i) whether the regression is carried out in a least-squares or in some other sense and (ii) whether the residuals (the departures of the specified data points from the model) are measured "vertically'' (i.e., in the "direction'' of the dependent variable), perpendicular to the response surface, or in some other way. Moreover, it is also necessary to define the "set of numerical values'', e.g., whether they form a column vector or a row vector (possibly a contiguous set of cells in a row or column in a spreadsheet model) or are defined by some other data structure.

An appropriate way to provide the specification is to define for the test software:

a) its inputs **x**,

b) its outputs **y**, and

c) the functional relationship or model that relates the inputs to the outputs. The model may take the form of an explicit formula specifying **y** in terms of **x**, viz.,

$$\mathbf{y} = \mathbf{f}(\mathbf{x}),$$

or an implicit relationship between ***x*** and ***y***, viz.,

$$\mathbf{g}(\mathbf{x}, \mathbf{y}) = \mathbf{0}.$$

If the specification is *inconsistent* with the task carried out by the software, testing in accordance with the specification would yield the conclusion that the software was deficient, when in fact it might be acceptable.

It is emphasised that the specification is a *mathematical* description of the task carried out by the test software. The specification is not intended to define the numerical methods used since these aspects are part of the solution implemented by the software. Thus, for the standard deviation example (see below), the use of formula (4) in the specification is equivalent to the use of formula (3). (However, it is helpful to give advice in the specification, e.g., that a particular formula or numerical method is as stable as possible, for implementation purposes.)

Example. Software for calculating the arithmetic mean and standard deviation may be specified in terms of inputs **x**, the set of numerical values $\{x_i: i = 1, \ldots, m\}$, outputs **y**, the arithmetic mean $\bar{x}$ and standard deviation s, and formulae

$$\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i, \quad s = \left\{\frac{1}{m-1}\sum_{i=1}^{m}(x_i - \bar{x})^2\right\}^{1/2}.$$

Example. Software for calculating the least-squares best-fit straight line to data may be specified in terms of inputs **x**, the data points $\{(x_i, f_i): i = 1, \ldots, m\}$, outputs **y**, the straight-line coefficients y_1 and y_2, obtained by solving the mathematical problem

$$\text{minimise} \quad \sum_{i=1}^{m}\{f_i - (y_1 + y_2 x_i)\}^2 \quad \text{with respect to } y_1, y_2.$$

2.2 Implementation of the test software

We suppose that the test software either

a) takes the form of a given software function, perhaps included as an in-built function within a commercial spreadsheet or other software package, or

b) is implemented using appropriately chosen (lower-level) in-built functions.

The two cases are intended to reflect the distinction between a *user* and a *developer* of the software. From the viewpoint of its testing, however, no distinction is necessary, since in either case the test software is regarded as a "black-box".

For the application of the testing methodology described in this work, the following practical considerations arise:

- how to generate reference data sets and results, and
- how to apply the test software to the reference data sets and subsequently evaluate the quality metrics and performance measures discussed below.

Generally, these operations will be undertaken using software although, for the former, reference data sets and results for calculations used commonly in metrology may be available from a reputable source. An issue is the management of the interface between the software for carrying out these operations and the test software. For a developer of the test software, the interface can often be made quite seamless. This is because the developer usually has a large amount of control over the software and can choose to test the software in isolation. For a user of the test software, the interface may be more cumbersome and involve writing and reading data files of reference data sets and results. In either case, it must be possible to

1. pass input data to the test software and extract the appropriate output results (this requirement is hopefully covered by the functional specification), and
2. pass input data and extract output results to a sufficient numerical precision (this requirement may also be covered by the functional specification).

It is necessary that the reference data are input to the test software as accurately as possible. Ideally, they should be input to a relative accuracy comparable to the computational precision η. If this is not done, the data used by the test software will be different from the intended reference data. As a consequence, the test software cannot be expected to perform so well, especially on ill-conditioned problems, with likely unnecessarily deleterious effects on the resulting performance measures. Furthermore, when comparing reference results and test results, spurious agreement may be obtained if the latter are output to a limited precision.

3. Design of Reference Data Sets

3.1 Specification of reference data sets

Some of the main considerations in designing reference data sets for testing the numerical accuracy of software are

1. the identification of possible deficiencies in the test software, such as those identified in Section 1,
2. the construction of data sets with known properties, e.g., their condition or “degree of difficulty”,
3. the need to mimic actual measurement data sets, i.e., the construction of data sets for which the reference results are known and which are “close to” the data sets used in a particular application area, and
4. the need to generate large numbers of reference data sets to ensure adequate coverage of the space of possible inputs to the test software.

These aspects are considered below.

3.1.1 Performance parameters

The testing of software using a small number of data sets can provide valuable information about the software. To be useful, the sets are chosen to

a) be representative of data sets for which the test software is intended (for example, as defined by its functional specification),

b) be as disparate as possible in terms of their locations in the space of possible data inputs, and

c) test particular paths through the software.

Such data sets provide "spot checks" of the test software, and are useful in the sense that the software is regarded as deficient it is fails to perform adequately on any one of them. They are less effective for black-box testing for which little can be assumed about the algorithms that have been implemented.

In many cases it is possible to "parametrise" the space of possible data inputs using (so-called) *performance parameters*, and an alternative is then to generate *sequences* of data sets for which, within a given sequence, different data sets correspond to different choices of a specified performance parameter (or parameters). These performance parameters are not to be confused with the model parameters that may be part of the functional specification: they can include the model parameters, but also parameters describing properties of the input data such as its signal-to-noise ratio and quantities controlling the number and distribution of the data inputs. The performance of the test software is investigated for such a sequence of data sets, in other words as a function of a performance parameter or parameters. In this way regions of the space of possible data inputs are identified for which the performance is adequate and regions where it is not. Moreover, having identified suitable performance parameters, it is often straightforward to generate automatically large collections of data sets, thus ensuring that the black-box testing undertaken is effective.

In some circumstances, a performance parameter can be interpreted as controlling the condition or "degree of difficulty" of the data set. In this case, the sequence of data sets becomes a *graded* sequence in which each data set in the sequence represents a problem that is more difficult than the previous member of the sequence.

Example. For the calculation of the sample standard deviation, we might choose to specify reference data sets using the following performance parameters:

- the sample mean $\bar{x}$,
- the sample standard deviation s,
- the number m of sample values.

In addition, the parameter $\bar{x} / s$, the inverse of the coefficient of variation, that expresses the condition of the problem of calculating the sample standard deviation of a data set (Section 1.2) may be used to define a graded sequence of reference data sets.

Example. For the calculation of the least-squares best-fit polynomial to measurement data $\{(x_i, y_i)\colon i = 1, \ldots, m\}$ we might choose to specify reference data sets using the following performance parameters:

- the degree n of the polynomial,
- the standard deviation σ of the measurement error,
- the mean $\bar{x}$ of the data abscissa values,
- the number m of measurement points.

3.1.2 Application-specific reference data sets

The purpose of testing software within the framework of the Software Support for Metrology programme is to check whether it is fit for purpose within the context of the National Measurement System. Accordingly, the reference data sets defined for this purpose are required to reflect as far as possible the sets that would be obtained and used in practical measurement.

The performance parameters introduced above help to satisfy this requirement: by assigning suitable values or ranges of values to the performance parameters that specify the reference data sets, it is possible to restrict the data sets used in the testing to those that are representative of the application area. Where the reference data is generated to include "random" structures, e.g., to simulate measurement error, it is possible to generate the data in such a way that these random structures are constrained to inherit (physical) properties: this aspect is described in Section 4.1.

Example. Reference data sets for testing software for straight-line linear regression may be required to:

a) relate to straight lines with an intercept between zero and one: the intercept is chosen as a performance parameter used to specify the data sets and is restricted to the range zero to one;

b) reflect a measurement system for which the standard deviation of the measurement errors is proportional to the value of the independent variable: the random structures in the data are constrained to inherit this property.

Example. Another example is the measurements produced by a roundness-measurement instrument for a manufactured component containing form error such as lobing. In order to test the software used to determine an associated feature (a Gaussian circle, for example) for such measurements, reference data sets should be prepared to mimic the principle character of such measurements, i.e., to include form error in the form of lobing, and for which the reference results are known to guaranteed accuracy.

3.2 Specification of performance measures and testing requirements

Quality metrics are used to quantify the performance of the test software for the sequences of reference data sets to which the test software is applied. Furthermore, by specifying the requirements of the user or developer on the test software in terms of these metrics, it is possible to assess objectively whether the test software is "fit for purpose".

Generally, the metrics measure the departure of the test results returned by the test software from the reference results. The departure may be expressed in terms of

1. the difference between the test and reference results, an *absolute* measure of accuracy,
2. the number of figures of agreement, a *relative* measure of accuracy, and
3. a *performance measure* that accounts for factors including the computational precision of the arithmetic used to generate the test and reference results and the condition of the problem.

However, other metrics may also be used, for example those that measure the resources required by the test software to generate a solution. These include:

a) memory requirements, and

b) execution time.

Furthermore, in situations where a number of test implementations are available, it may be useful to consider metrics that measure the difference between the test results returned by these various implementations as well as to report their accuracy compared with reference results.

3.2.1 Absolute measures of accuracy

Suppose the reference results and test results are expressed as vectors of floating-point numbers. Let

$$\Delta\mathbf{y} = \mathbf{y}^{(\mathrm{test})} - \mathbf{y}^{(\mathrm{ref})},$$

where $\mathbf{y}^{(\text{test})}$ denotes the test results and $\mathbf{y}^{(\text{ref})}$ the reference results corresponding to reference data **x**. Then,

$$d(\mathbf{x}) = RMS(\Delta\mathbf{y}) \tag{8}$$

is an absolute measure of departure of the test results from the reference results for the reference data **x**. Here,

$$RMS(\mathbf{v}) = \frac{\|\mathbf{v}\|_2}{\sqrt{n}}$$

denotes the root-mean-square value of an *n*-vector **v**. For a scalar-valued result *y* or when applied to a single component of a vector-valued result, the measure reduces to

$$d(\mathbf{x}) = \left| y^{(\text{test})} - y^{(\text{ref})} \right|. \tag{9}$$

In cases where the components of **y** are not comparably scaled, some care is required when interpreting the summary measure $d(\mathbf{x})$ defined by (8), and consideration of the component-wise errors (9) is recommended.

3.2.2 Relative measures of accuracy

Let $d(\mathbf{x})$ be defined by (8) or (9). Then, the number of figures of agreement $N(\mathbf{x})$ between the test results and reference results corresponding to reference data **x** is calculated from:

If $d(\mathbf{x}) \neq 0$,

$$N(\mathbf{x}) = \min\left\{ M(\mathbf{x}), \log_{10}\left(1 + \frac{RMS(\mathbf{y}^{(\text{ref})})}{d(\mathbf{x})} \right) \right\}. \tag{10}$$

If $d(\mathbf{x}) = 0$,

$$N(\mathbf{x}) = M(\mathbf{x}).$$

Here, $M(\mathbf{x})$ is the number of correct significant figures in the reference results corresponding to **x**, and is included to account for the fact that the reference results may themselves have limited precision that reflects the condition of the problem.

As for the absolute measures of accuracy described in Section 3.2.1, (10) may be applied to the complete vector **y** or its components.

3.2.3 Performance measures

In Appendix A we derive the performance measure $P(\mathbf{x})$, where

$$P(\mathbf{x}) = \log_{10}\left(1 + \frac{1}{\kappa(\mathbf{x})\eta} \frac{\|\Delta\mathbf{y}\|}{\|\mathbf{y}^{(\text{ref})}\|} \right), \tag{11}$$

corresponding to reference data **x**. The measure is a quality metric that quantifies the performance of the test software and accounts for the following factors:

a) the difference $\Delta\mathbf{y}$ between the test and reference results,

b) the condition κ of the problem, and

c) the computational precision η.

It indicates the number of figures of accuracy *lost* by the test software over and above what software based on an optimally stable algorithm would produce, being near zero if $\mathbf{y}^{(\text{test})}$ and $\mathbf{y}^{(\text{ref})}$ agree as closely as could be expected, and having a value of about *p* if the agreement is *p* figures less than this. An optimally-stable algorithm would produce as much accuracy as is possible in solving the problem defined by the specification and the data set, using the available

computational precision. A related performance measure is used in testing software for evaluating special functions [3].

In Appendix B we consider the determination of a performance measure in circumstances in which reference values of less than ideal accuracy are available. It is shown that a sensible estimate of the performance measure can be made by making some assumptions about an available approximation to the reference values.

Example. For the calculation of the sample standard deviation *s* corresponding to a set of values $\mathbf{x} = \{x_i: i = 1, \ldots, m\}$,

$$P(\mathbf{x}) = \log_{10}\left(1 + \frac{1}{\kappa(\mathbf{x})\eta}\frac{|\Delta s|}{|s|}\right),$$

where (see Appendix A.2)

$$\kappa^2(\mathbf{x}) = \left(\frac{m-1}{m}\right)^2 + \frac{m-1}{m}\left(\frac{\bar{x}}{s}\right)^2,$$

and $\bar{x}$ is the sample arithmetic mean.

Example. For the calculation of the residuals **r** corresponding to the solution of the least-squares regression

$$\min_{\mathbf{b}} \|\mathbf{y} - A\mathbf{b}\|^2,$$

it is shown in Appendix A.3 that

$$P(\mathbf{y}) = \log_{10}\left(1 + \frac{\|\Delta\mathbf{r}\|}{\|\mathbf{y}\|\eta}\right).$$

4. Use of Reference Data Sets

4.1 Generation of reference pairs

A *reference pair*, i.e., a reference data set and the corresponding reference results, may be produced in two ways:

1. Start with a reference data set and apply *reference software* (as, e.g., in [8]) to it to produce the corresponding reference results.
2. Start with some reference results and apply a *data generator* to them to produce a corresponding reference data set.

Procedures for undertaking software testing according to 1 and 2 above are illustrated in Figures 1 and 2, respectively. In the former, the input to the procedure is a reference data set whereas for the latter the input takes the form of reference results.

The approach adopted here is to use a data generator to construct reference data sets to have known solutions, i.e., solutions specified *a priori*. Data generators can generally be implemented for problems for which a mathematical *characterisation* of the solution to the problem exists, and this applies to a wide range of calculations used in metrology.

The alternative approach described in 1 above relies on the availability of reference software, i.e., software written to an extremely high standard to solve the problem given in the functional specification. It is akin to a *primary standard* in measurement, such as a kilogram mass to which secondary standards are compared for calibration purposes. It has been stated [1], however, that reference software is very demanding to provide. At the National Physical Laboratory we have found that the effort required to produce a data generator can be a small

fraction of that to produce reference software. The reason is that the production of a data generator tends to be a much more compact operation, with fewer numerical pitfalls, and hence its testing overhead is significantly less than that of reference software for the forward calculation (i.e., of determining reference results from reference data sets).

For problems with a unique solution there is one set of reference results corresponding to a given reference data set. Conversely, for given reference results there is in general an infinite number of corresponding reference data sets. This latter property can be used to considerable advantage in generating reference data sets, both in terms of forming data sets having selected condition numbers or "degrees of difficulty", and in determining data sets which mimic actual data sets from applications. As a simple example, there is a unique sample standard deviation *s* for a given sample of two or more numbers, whereas given *s* there are infinitely many data sets having this value as their standard deviation.

Null-space methods [14, 15] provide a facility for generating *families* or *classes* of data sets, and are the basis of many of the data generators described here. When any of these sets is submitted to correctly working software, nominally the same solution is produced in each case. Moreover, this solution is identical to a known solution. A further advantage of null-space methods is that they can be used to characterise the *space of reference data sets* having the given solution. In particular, a sequence of data sets from this space can be extracted having a range of degrees of difficulty that proves to be invaluable for software testing.

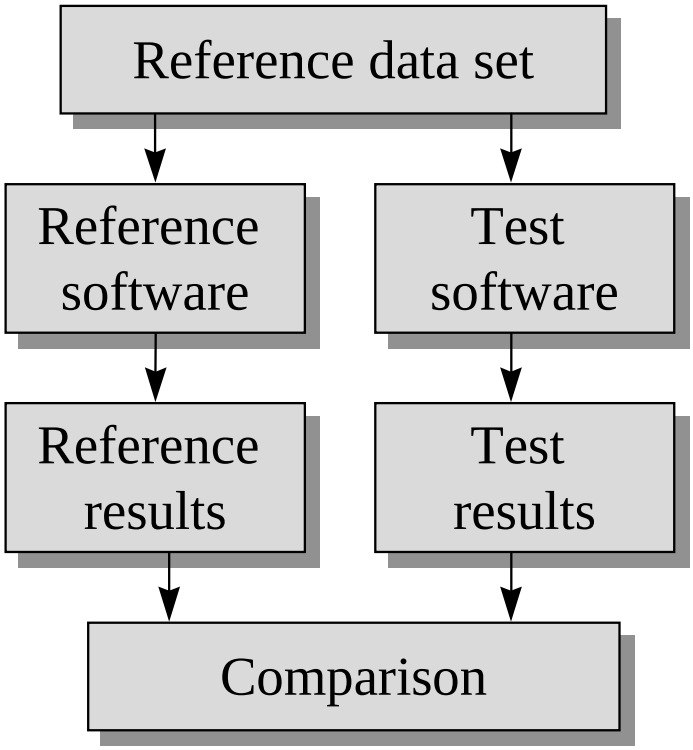

Figure 1 Procedure for testing software using reference software.

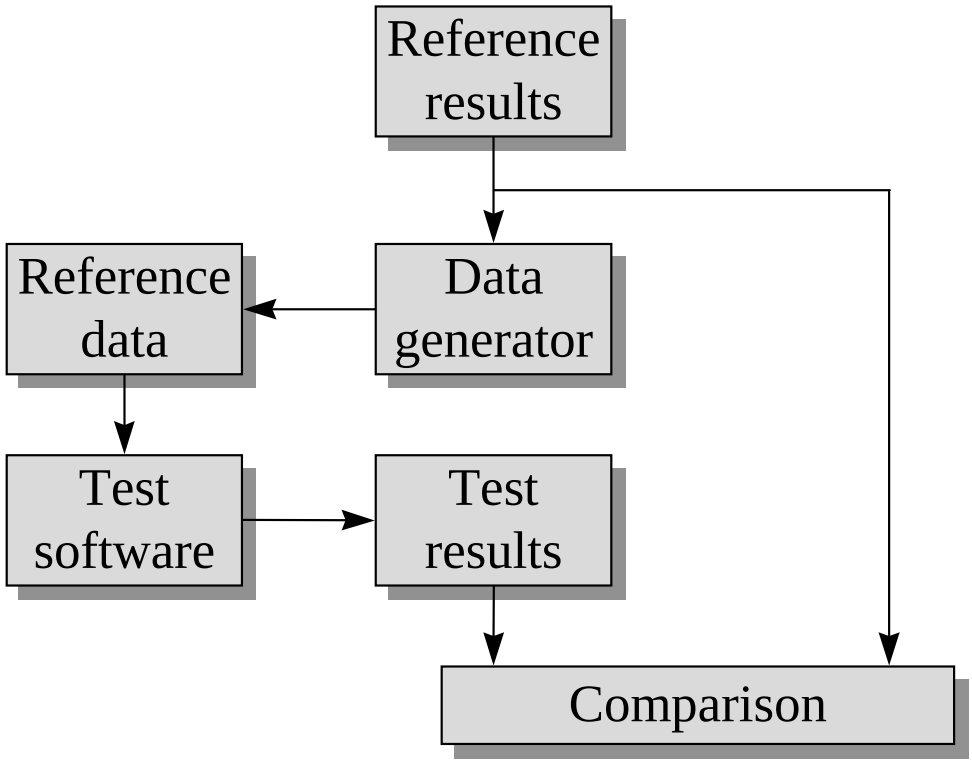

Figure 2 Procedure for testing software using data generators.

Example. The null-space approach is now illustrated for the linear least-squares model

$$\mathbf{y} = A\mathbf{b} + \mathbf{r}, \tag{12}$$

where A denotes the observation matrix, $\mathbf{y}$ the observation vector, $\mathbf{b}$ the vector of model parameters and $\mathbf{r}$ the vector of residuals. For example, if the data to be fitted is $\{(x_i, y_i)\colon i = 1,\ldots, m\}$, and the model is the straight line with parametrisation (1), then

$$A = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \quad \mathbf{r} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}.$$

Now the least-squares solution is characterised by

$$A^{\mathrm{T}}\mathbf{r} = \mathbf{0} \tag{13}$$

(see [10]). This equation implies two conditions, one that

$$\sum_{i=1}^{m} r_i = 0,$$

i.e., the sum of the residuals is zero, and the other that

$$\sum_{i=1}^{m} x_i r_i = 0,$$

i.e., the first moment (with respect to the abscissa values) of the residuals is zero. Let N be a basis for the null space of A^{T}, i.e.,

$$A^{\mathrm{T}} N = \mathbf{0}.$$

This basis can be expressed as an independent set of (column) vectors which together form N. This set is in general not unique. It is usually determined to be orthogonal, i.e., the inner product of any two (different) vectors is zero. A vector in the null space can be represented as a linear combination of these "null" vectors. Then, for a vector

$$\mathbf{r} = N\mathbf{u},$$

for *any* choice of vector $\mathbf{u}$, the replacement of $\mathbf{y}$ by $\mathbf{y} + \mathbf{r}$ will leave the solution vector $\mathbf{b}$ *unchanged*. This result can be seen as follows. The condition (13) and the model (12) imply the *normal equations*

$$A^{\mathrm{T}} A\mathbf{b} = A^{\mathrm{T}}\mathbf{y}.$$

But, from the definition of the null space,

$$A^{\mathrm{T}}(\mathbf{y}+\mathbf{r}) = A^{\mathrm{T}}\mathbf{y} + A^{\mathrm{T}} N\mathbf{u} = A^{\mathrm{T}}\mathbf{y}.$$

Thus, from any one data set any number of further data sets having the same solution can readily be constructed by choosing vectors $\mathbf{u}$.

A procedure for generating a reference data set $\mathbf{y}$ given the observation matrix A and model parameters $\mathbf{b}$ then takes the following form:

I. Evaluate the observation vector $\mathbf{y}_0$ given by $\mathbf{y}_0 = A\mathbf{b}$.

II. Form null space matrix N for A^{T}.

III. Form residual vector $\mathbf{r}$ given by $\mathbf{r} = N\mathbf{u}$, where the elements of $\mathbf{u}$ are chosen to be random numbers. (Often, $\mathbf{r}$ will be scaled appropriately to represent actual measurement error, or other constraints may be applied: see below.)

IV. Form observation vector $\mathbf{y}$ given by $\mathbf{y} = \mathbf{y}_0 + \mathbf{r}$.

The problem of reference software has hence essentially been "replaced" by that of a reference implementation of "null", which is at the heart of data generation for *all* regression problems. The null space of a *general* matrix A can be calculated very reliably using the singular-value decomposition [10, p602]. One such implementation is the `null` function in Matlab [16]. Additionally, it is possible to check the accuracy of the generated data sets by forming $A^{\mathrm{T}}\mathbf{r}$ and assessing the closeness of $\|A^{\mathrm{T}}\mathbf{r}\|_2$ to zero, relative to the value of $\|A^{\mathrm{T}}\|_2\|\mathbf{r}\|_2$. This check is generic and does not require the availability of software to solve the particular problem considered.

Now suppose we wish to construct reference data for which the residuals $\mathbf{r}$ are "close" to specified values $\mathbf{r}_0$. For example, to mimic a measurement process in which the residual errors depend on the corresponding abscissae values, the element $r_{i,0}$ of $\mathbf{r}_0$ is assigned to be a sample taken from a Gaussian distribution with mean zero and standard deviation σx_i for a given σ that controls the overall size of measurement error.

The problem is to choose an element of the null-space that is close to $\mathbf{r}_0$, i.e., to solve

$$\min_{\mathbf{u}} \|\mathbf{r}_0 - N\mathbf{u}\|.$$

But the formal solution to this problem satisfies the normal equations

$$N^{\mathrm{T}} N\mathbf{u} = N^{\mathrm{T}}\mathbf{r}_0,$$

and since N is orthogonal, i.e., $N^{\mathrm{T}}N = I$, the identity matrix, so

$$\mathbf{u} = N^{T}\mathbf{r}_0.$$

4.2 Presentation and interpretation of performance measures

Having applied the test software to a reference data set to give a test result, the test result is compared with reference results corresponding to the data set by computing one or more of the

quality metrics described in Section 3.2. If a sequence of data sets is available corresponding to a range of values of a performance parameter, the quality metrics may be computed as functions of this parameter. The quality metrics are presented either

a) in tabular form, or

b) as a graph plotted against the performance parameter.

It is also convenient to give summary statistics for the computed quality metrics, including their arithmetic means, sample standard deviations, minima and maxima in order to give insight into how the performance of the test software depends on the performance parameter.

A graph of the values of the performance parameter $P(\mathbf{x})$ plotted against the problem condition $\kappa(\mathbf{x})$ (or some related performance parameter), can provide valuable information about the software. The resulting performance profile can be expressed as a series of points (or the set of straight-line segments joining them). By introducing variability into each data set for each value of the performance parameter, e.g., by using random numbers in a controlled way to simulate observational error, the performance profile will become a *swathe* of "replicated'' points, which can be advantageous in the fair comparison of rival algorithms. Examples of performance profiles are given in Section 6, as well as in [6, 7]. Performance profiles in a somewhat different setting have also been studied by Lyness [17].

Fitness for purpose implies that the test software meets requirements as specified using quality metrics or other appropriate measures (see Section 3.2). The user should verify whether the reference data set(s) used can be regarded as sufficiently representative of the application. It is then necessary to decide whether absolute or relative measures of accuracy are relevant. If absolute errors are important, the user should decide whether the value of $d(\mathbf{x})$, the absolute error between the test and reference results, given by equation (8) or (9) is acceptably small. If relative errors are important, the user should decide whether the value of $N(\mathbf{x})$, the number of figures of agreement between the test and reference results, given by equation (10), is acceptably small.

The performance measure $P(\mathbf{x})$ given by (11) indicates the number of figures of accuracy lost by the test software in the computing the test results compared with the reference results. A large value may indicate the use of an unstable parametrisation of the problem, the use of an unstable algorithm or inappropriate formula, or that the test software is defective. If for graded reference data sets taken in order the corresponding performance measures $P(\mathbf{x})$ have a tendency to increase, it is likely that an unstable parametrisation, formula or numerical method has been employed.

5. Other Tests of Numerical Accuracy

5.1 Other uses of reference data sets

The above tests are concerned with the assessment of test software against an absolute, i.e., against what optimal software would be expected to produce. Alternatively, it is possible to carry out a *comparative* assessment of two or more items of software which purportedly carry out the same function.

Instead of producing a performance profile, showing how one or more items of test software compare against the best possible, a graph illustrating the performance of Test Software A relative to Test Software B is produced. For example, in addition to tabulating the departure of the results of Test Software A from the reference results, these departures can be compared numerically and visually with those for Test Software B.

In addition, a relative performance measure for the two competing test software implementations may be derived. Recalling the approximation

$$\log(1+X) \approx \log X$$

which is valid provided $X >> 1$, we make the following approximations

$$P_A(\mathbf{x}) = \log_{10}\left(1+\frac{1}{\kappa(\mathbf{x})\eta}\frac{\|\Delta\mathbf{y}_A\|}{\|\mathbf{y}^{(\mathrm{ref})}\|}\right) \approx \log_{10}\left(\frac{1}{\kappa(\mathbf{x})\eta}\frac{\|\Delta\mathbf{y}_A\|}{\|\mathbf{y}^{(\mathrm{ref})}\|}\right)$$

and

$$P_B(\mathbf{x}) = \log_{10}\left(1+\frac{1}{\kappa(\mathbf{x})\eta}\frac{\|\Delta\mathbf{y}_B\|}{\|\mathbf{y}^{(\mathrm{ref})}\|}\right) \approx \log_{10}\left(\frac{1}{\kappa(\mathbf{x})\eta}\frac{\|\Delta\mathbf{y}_B\|}{\|\mathbf{y}^{(\mathrm{ref})}\|}\right),$$

(equation (11)) which are valid provided the implementations are not optimal. Then, defining

$$P_B^A(\mathbf{x}) = P_A(\mathbf{x}) - P_B(\mathbf{x}),$$

we obtain

$$P_B^A(\mathbf{x}) = \log_{10}\left(\frac{\|\Delta\mathbf{y}_A\|}{\|\Delta\mathbf{y}_B\|}\right). \qquad (14)$$

(Here, to ensure that $P_B^A(\mathbf{x})$ always takes a finite value, the difference between a test result and the reference result is replaced by $\eta\|\mathbf{y}_A\|$ or $\eta\|\mathbf{y}_B\|$whenever that difference is computed as zero.) The performance measure indicates the number of figures of accuracy *lost* by Test Software A compared with Test Software B. Because it does not reflect the *condition* of the problem of computing the reference result $\mathbf{y}^{(\mathrm{ref})}$ or the computational precision of the arithmetic used to generate the test results, it does not provide an absolute measure of performance. However, it does enable the *ranking* of test software implementations in the sense that $P > 0$ indicates that Test Software B returns a more accurate value than Test Software A, and $P < 0$ indicates that Test Software A returns a more accurate value than Test Software B.

5.2 Tests not requiring reference data sets

Examples of tests which do not require the availability of reference results are

1. consistency checks,
2. continuity checks,
3. spot checks against tabulated or other values, and
4. checking solution characterisations.

All such checks are to be regarded as *complementary* to and not alternatives to the main tests.

5.2.1 Consistency checks

An example of consistency checking is the use of a forward followed by an inverse calculation, or *vice versa*. For example, given test software for sin *x* and test software for $\sin^{-1} x$, within the same package under test, the values of $\sin^{-1}\{\sin x\}$ can be compared with *x* for a range of values of *x*. It is essential to observe that this form of checking alone is insufficient. Two functions could in principle form a near-perfect "inverse pair" but apply to a different mathematical operation.

5.2.2 Continuity checks

The algorithms underpinning much mathematical software for special functions such as Bessel functions utilise several mathematical approximations (Chebyshev series, rational functions, etc.), each of which is valid over sub-ranges of the argument(s). By evaluating the test software which implements such a function over suitable ranges it is possible to gauge the extent to

which the returned values are continuous across sub-range boundaries. The presence of discontinuities can have a damaging effect on software which uses these functions as modules, e.g., in adaptive quadrature and optimisation. The documentation accompanying such software may sometimes indicate the sub-range boundaries. Otherwise, they may be inferred from the testing process by graphing the results for a dense set of input-argument values.

5.2.3 Spot checks against tabulated or other values.

For certain arguments or ranges of the argument it may be possible to compare the results produced by test software with tabulated values or the results from other software. As an example, functions of a complex variable can be reduced to real functions if the argument is purely real or purely imaginary or through the use of mathematical identities, e.g.,

$$e^{ix} = \cos x + i \sin x$$

and related formulae. In such cases alternative software may exist which is already well characterised.

5.2.4 Checking solution characterisations

The approach adopted in this work to the generation of reference data sets and corresponding results is to use a data generator. It has been stated (Section 4.1) that data generators can generally be implemented for problems for which mathematical characterisation of the solution to the problem exists. A data generator for the least-squares regression problem has been described in Section 4.1, and a further example is given in Section 6.3.

In addition to their use as the basis of data generators, solution characterisations can be valuable for checking directly the results returned by test software [18]. For example, suppose $\mathbf{r}_t$ is the residual vector returned by test software for the least-squares regression problem (12), and define

$$\mathbf{e} = A^{\mathrm{T}}\mathbf{r}_t .$$

Then, recalling the characterisation (13), if $\mathbf{e} = \mathbf{0}$ then $\mathbf{r}_t$ is correct. If $\mathbf{e}$ is small (in an appropriate sense), $\mathbf{r}_t$ can be taken as an acceptable solution. The approach requires no knowledge of the true solution $\mathbf{r}$. Instead, we rely on the fact that it is possible to test the properties the solution is supposed to possess using the given characterisation. Such tests constitute *post-processing* tests of the software and can be applied using simulated test data or data arising in the real world.

6. Examples of Application

In this section we give a number of examples of the application of the testing methodologies described before.

We start with some examples of statistical calculations that are used in metrology. The sample standard deviation is an important summary statistic used, for example, to quantify the repeatability of a measurement process. In Section 6.1 we show how reference data sets are used to distinguish between algorithms implementing the two mathematically equivalent formulae for the statistic given in Section 1.2. Principal-components analysis (PCA) is a tool used as part of the process of model building in regression analysis. The calculations underpinning PCA are more complex than that required for the sample standard deviation. Nevertheless, we show in Section 6.2 how reference data sets are used to distinguish between numerical algorithms implemented in software for PCA.

Regression analysis forms the basis of much of the processing undertaken of metrology data. We have already indicated a procedure for generating reference data sets for linear least-squares regression based on the so-called null-space approach (Section 4.1). In [6] we consider

in some detail the application of the testing methodology to a non-linear least-squares regression problem in which the model takes the form of a single Gaussian peak. In order to demonstrate that we construct data generators for other types of regression problems for which, for example, the null-space approach is not appropriate, we consider in Section 6.3 the linear minimax or Chebyshev regression problem.

Finally, in Section 6.4, we give an example of how simple continuity checks (Section 5.2.2) are used to identify numerical properties of a simple calculation involving complex-valued quantities.

6.1 The sample standard deviation

Consider a basic data set defined by $X_0 = [\mu - nh, \mu - (n - 1)h, \ldots, \mu + nh]$. The arithmetic mean and standard deviation of this data set are μ and $s = h\{(n + 1/2)(n + 1)/3\}^{1/2}$, respectively. A family of graded data sets can be derived from this set by adding an increasing sequence of values: $X_k = X_0 + q^k$, $k = 1, \ldots, N$, for some $q > 1$. The mean and standard deviation of X_k are $\mu + q^k$ and s, respectively. Graded data sets were so constructed using $\mu = 3.172$, $h = 0.1$, $q = 1.5$, $n = 12$ and $N = 50$. A performance parameter was calculated in each case as the inverse coefficient of variation $\bar{x} / s$ [1]. Software implementations of the formulae (3) and (4) for calculating the standard deviation were applied to these data sets and a performance profile determined (Section 3.2.2). These performance profiles are illustrated in Figure 3.

It is observed in Figure 3 that the stable algorithm (points joined by straight-line segments to give an essentially horizontal line) performs completely satisfactorily for all the data sets. The unstable algorithm (points joined by straight-line segments to give a generally increasing line) loses a number of decimal figures over and above that which would be lost by a stable algorithm in a manner essentially proportional to $\log(\bar{x} / s)$. (This result is predicted by a detailed floating-point error analysis of formula (4).) A performance profile for standard-deviation software similar in behaviour to the generally rising line in Figure 3 could be taken as an indication that the unstable formula (4) (or a comparable formula) had been used. It is to be noted that, for almost 15% of the 50 cases, the results produced by the unstable algorithm are as accurate as those for the stable algorithm. This demonstration of the fact that an unstable algorithm is (almost randomly) capable of providing good results is a reason why minimal checking of a piece of software is unsatisfactory.

There are counterparts of this result for many other mathematical and statistical calculations.

It is worth emphasising here the dependence of the condition of the problem on the value of the standard deviation, and in particular how the problem becomes increasingly ill-conditioned as the standard deviation decreases provided the arithmetic mean remains fixed. The consequence of this is that if the data is obtained from a measurement process that is made more accurate, in the sense of reducing the “true” standard deviation of the data, the problem of calculating accurately that standard deviation is made more difficult. The necessity of using high-quality software for the calculation is therefore greater. This has implications for high-accuracy and primary metrology as is frequently undertaken in the National Measurement System programmes. It is in distinct contrast to many other areas where the data has greater scatter: for such data the need for high-quality software is less critical.

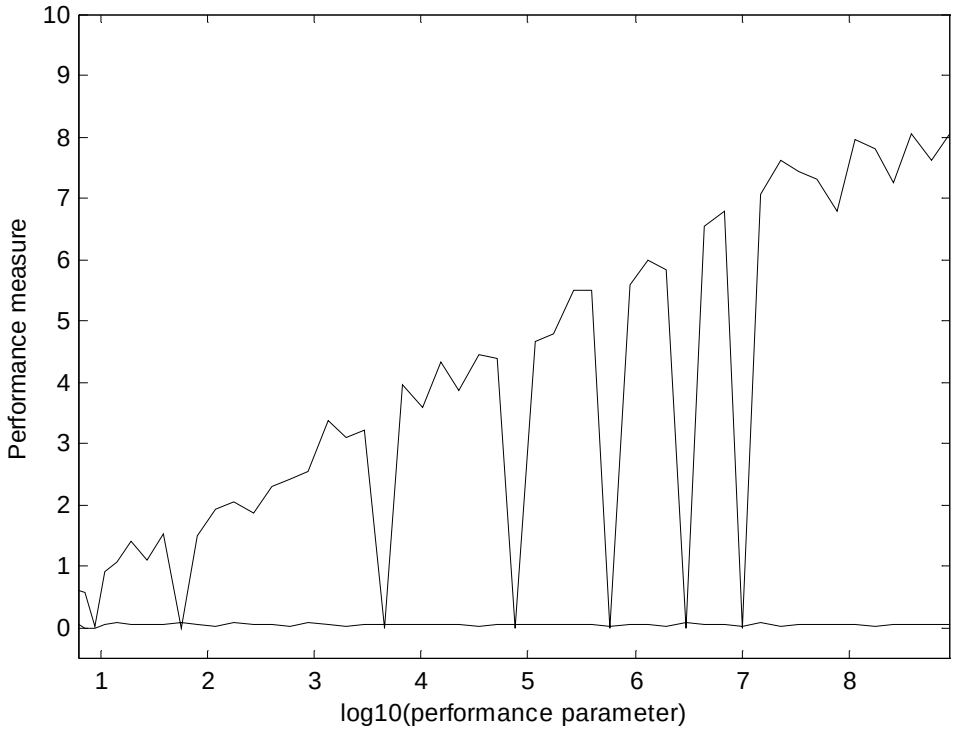

Figure 3 Performance profile for sample standard deviation against inverse coefficient of variation calculated using a stable algorithm (generally-horizontal line) and an unstable algorithm (generally-increasing line).

6.2 Principal components analysis

In building a good calibration model in analytical chemistry, a key step is to predict concentration levels of the analyte of interest, by assembling a representative range of samples [19]. Principal-components analysis (PCA) is a tool that assists in determining this model. It is valuable because of its ability to reduce the number of variables in a model to the minimum number that provide the required information. Mathematically, the PCA problem is as follows. Given an $n \times p$ observation matrix X, determine a number, $l < p$, say, of linear combinations $\mathbf{y}_j$ of the columns $\mathbf{x}_j$ of X such that the $\mathbf{y}_j$ are uncorrelated and have maximal variance amongst all such linear combinations. The $\mathbf{y}_j$ are the columns of the matrix Y obtained from X by the column transformation $Y = XV_l$, where V_l is a $p \times l$ matrix whose columns are the normalised eigenvectors associated with the l most significant eigenvalues λ_i of the normal matrix $H = X^TX$ [20].

There are two predominant algorithms for solving the PCA problem. One involves forming $H = X^TX$ explicitly and employing eigendecomposition software: $H = VDV^T$ [10, p391]. Here, D is the diagonal matrix of order p whose diagonal entries are the eigenvalues of H and V is the (orthogonal) matrix of order p whose columns are the corresponding eigenvectors. (Advantage can be taken algorithmically of the fact that H is symmetric.) The other algorithm forms the singular-value decomposition (SVD) [10, p69] of X: $X = USV^T$, where S is the diagonal matrix of order p whose diagonal entries are the singular values of X, V is a matrix of order p containing the corresponding right singular vectors and U is an orthogonal matrix of order n.

The above multiple use of the symbol V is deliberately suggestive. Mathematically, the V obtained in both decompositions is the same and $D = S^2$. These results are immediately obtained from the fact that $X = USV^T$ implies that $H = X^TX = VS^2V^T$, since U is orthogonal. Moreover, $V = V_p$.

Numerically, the SVD is preferable to eigendecomposition because the latter can be expected to lose more figures than the former, consequent on working with *H* rather than directly with *X*. If, furthermore, *X* (and hence *H*) has close singular values, both eigendecomposition and the SVD will inevitably suffer further loss of accuracy, which could be disastrous for the former method. If *X* has close singular values, this additional loss can be considerable.

To illustrate, we use an example from [1]. Consider a 9×6 matrix *X* which is

$$\begin{pmatrix} 0.00748716773088 & 0.00951114748716 & -0.00608079406884 & 0.00581067039528 & 0.00567196848036 & 0.11339287808508 \\ -0.00748906373088 & -0.00951136948716 & -0.00608477206884 & -0.00581454639528 & -0.00567513048036 & -0.11339216408508 \\ -0.05738659687008 & -0.07292893560156 & 0.04663290299444 & -0.04455478863048 & -0.04349024388276 & -0.86934423525228 \\ 0.03279479065440 & 0.04167680143580 & -0.02664530634420 & 0.03809106397640 & 0.02537218640180 & 0.56763902242540 \\ -0.06688083740832 & -0.08499449762724 & 0.05434015785676 & -0.05229965383992 & 0.03425494607796 & -0.63899102008212 \\ -0.00156648314112 & -0.00199116289184 & 0.00127055445216 & 0.00150296491328 & -0.02113433072864 & -0.09729521564192 \\ 0.12725435222496 & 0.16171937288172 & -0.10339259357028 & 0.09880000151976 & 0.09643864176612 & 1.92767550024636 \\ -0.06237958442400 & -0.07927420239300 & 0.05068264390700 & -0.04843137329400 & -0.04727384400300 & -0.94493897070900 \\ 0.05738921767008 & 0.07293226620156 & -0.04662803239444 & 0.04455686343048 & 0.04349193648276 & 0.86934385305228 \end{pmatrix}$$

to 14 decimal places. Because of the way in which the data was generated the corresponding matrix of eigenvectors is exactly

$$V = \begin{pmatrix} 0.5392 & -0.5856 & 0.3744 & -0.3648 & -0.2976 & 0.0672 \\ 0.5392 & 0.2558 & 0.4758 & -0.4636 & -0.3782 & 0.0854 \\ 0.3744 & 0.4758 & 0.6958 & 0.2964 & 0.2418 & -0.0546 \\ -0.3648 & -0.4636 & 0.2964 & 0.7112 & -0.2356 & 0.0532 \\ -0.2976 & -0.3782 & 0.2418 & -0.2356 & 0.8078 & 0.0434 \\ 0.0672 & 0.0854 & -0.0546 & 0.0532 & 0.0434 & 0.9902 \end{pmatrix}.$$

X has some close eigenvalues. The complete eigenvalue spectrum is (exactly)

$$\{0.000009, 0.000036, 0.000049, 161.29000, 6756.84000, 7023030.01000\}.$$

Even the best-possible algorithm will lose a certain number of figures in determining *V*, since the eigenvalues are close relative to their range; this is less true of the corresponding singular values.

We find that software implementing the SVD approach to PCA delivered a *V*-matrix which differed in 2-norm from the true *V* by 2×10^{-12}. This result is close to best possible for the computational precision η and the condition of the problem. For eigendecomposition-based software the *V*-matrix computed differed in 2-norm from the true *V* by 2×10^{-7}. Thus, it can be concluded that the use of an eigendecomposition algorithm loses five more figures than does the SVD *for this matrix*. For other matrices, the loss may be greater or less.

By determining a graded set of 9×6 matrices, i.e., a sequence whose condition numbers successively increase, a performance profile as in Section 6.1 can be produced. As before, if there is a tendency for the performance measure to increase with the performance parameter, this is an indication that an unstable algorithm (probably in this case based on an eigendecomposition) has been used, rather than the SVD which would yield an essentially horizontal performance profile. The results so obtained can be repeated with matrices of other dimensions to help confirm any conclusion drawn about the quality of the test software.

6.3 Linear minimax regression

Least-squares or l_2 regression is probably the most widely-used type of regression for metrology applications. Where data errors are samples of a Gaussian probability distribution, as is often assumed for measurement noise, it gives estimates that have good statistical properties in terms of low bias and high efficiency. However, situations arise where the measurement errors are such that alternative types of regression are appropriate. For example,

a) the l_1 norm ("min-sum-mod") is appropriate when the data errors are distributed exponentially, i.e., have a long-tailed distribution, and is thus useful when wild points (outliers) are likely in the measurement data, and

b) the l_∞ norm (minimax or Chebyshev) is used when the data errors are distributed uniformly as arises, for example, when the measurement data are given by a digital readout and all indicated figures can be regarded as exact.

Mathematical characterisations for the solutions to regression problems based on the l_1 and l_∞ norms may be derived. We indicate below the characterisation for a solution to the linear minimax regression problem, and show how this characterisation may be used to construct a data generator. Unlike the corresponding least-squares problem, the data generator is not based on the null-space approach. We have used at the National Physical Laboratory reference data sets produced in this way to test an algorithm for linear minimax regression incorporated in a software package for computing polynomial and polynomial spline calibration curves for experimental data.

Recall that the linear *least-squares* regression problem may be written in the forms

$$\min_{\mathbf{b}} \|A\mathbf{b} - \mathbf{y}\|_2^2 \equiv \min_{\mathbf{b}} \sum_{i=1}^{m} \left|\mathbf{a}_i^{\mathrm{T}}\mathbf{b} - y_i\right|^2,$$

where y_i, $i = 1, \ldots, m$, is the given data, A is the regression or observation matrix made up of row vectors $\mathbf{a}_i^{\mathrm{T}}$, $i = 1, \ldots, m$, and $\mathbf{b}$ is the vector of regression parameters to be determined. In least-squares regression the objective function to be minimised is the sum of squares of the residual errors. The corresponding linear *minimax* regression problem is to solve the following optimisation problem:

$$\min_{\mathbf{b}} \max_{i=1,\ldots,m} \left|\mathbf{a}_i^{\mathrm{T}}\mathbf{b} - y_i\right|, \tag{15}$$

where y_i, $\mathbf{a}_i^{\mathrm{T}}$ and $\mathbf{b}$ have the meanings given above. In minimax regression the objective function to be minimised is the residual error of maximum absolute value.

Defining

$$\zeta = \max_{i=1,\ldots,m} \left|\mathbf{a}_i^{\mathrm{T}}\mathbf{b} - y_i\right|,$$

an equivalent problem is to solve

$$\min_{\mathbf{b},\zeta} \zeta \qquad \text{subject to} \qquad \begin{cases} \zeta - (\mathbf{a}_i^{\mathrm{T}}\mathbf{b} - y_i) \geq 0, \\ \zeta + (\mathbf{a}_i^{\mathrm{T}}\mathbf{b} - y_i) \geq 0, \end{cases} \quad i = 1, \ldots, m.$$

This equivalent problem, in which the unknown parameters are $\mathbf{b}$ and ζ, is more convenient for analysis because it involves smooth functions. Furthermore, the problem is now in the form of a linear program, i.e., both the objective function and the constraints are linear functions of the unknown parameters.

Necessary and sufficient conditions for $\mathbf{b}^*$ and ζ^* to define a solution to this linear program are given in [9]. These conditions require that $(\mathbf{b}^*, \zeta^*)$ is feasible and that certain conditions involving the constraints that are active at $(\mathbf{b}^*, \zeta^*)$ are satisfied. These conditions are given below.

Let $(\mathbf{b}^*, \zeta^*)$ be a *feasible* point, i.e., one for which all the constraints are satisfied. Furthermore, let I^+ and I^- be (mutually exclusive) subsets of $I = \{i: i = 1, \ldots, m\}$ containing the indices of the constraints that are *active*, i.e., are satisfied exactly with

$$\begin{aligned} \zeta^* - (\mathbf{a}_i^{\mathrm{T}}\mathbf{b}^* - y_i) = 0, \quad i \in I^+, \\ \zeta^* + (\mathbf{a}_i^{\mathrm{T}}\mathbf{b}^* - y_i) = 0, \quad i \in I^-. \end{aligned}$$

Then (see [9]), sufficient conditions for $(\mathbf{b}^*, \zeta^*)$ to be a solution to the linear minimax regression problem defined by (15) are that there exist (Lagrange) multipliers for which

$$\sum_{i\in I^+}\lambda_i\mathbf{a}_i=\sum_{i\in I^-}\mu_i\mathbf{a}_i, \tag{16}$$

that satisfy

$$\begin{aligned}\lambda_i&>0,\quad i\in I^+,\\ \mu_i&>0,\quad i\in I^-,\end{aligned} \tag{17}$$

and

$$\sum_{i\in I^+}\lambda_i+\sum_{i\in I^-}\mu_i=1. \tag{18}$$

A procedure for generating a reference data set for the linear minimax regression problem based on the characterisation (17)–(18) is given below. The output of the procedure is a set of vectors $\mathbf{a}_i$ and a set of observations y_i for which the solution to the problem defined by (15) is known to be $\mathbf{b}^*$ with minimax error ζ^*.

I Choose n^+ Lagrange multipliers λ_i, $i\in I^+$, and n^- Lagrange multipliers μ_i, $i\in I^-$, to be strictly positive and to sum to unity, i.e., to satisfy (17) and (18).

II Choose n^+ vectors $\mathbf{a}_i$, $i\in I^+$, and n^- vectors $\mathbf{a}_i$, $i\in I^-$, to satisfy (16). This can be done by choosing all the vectors $\mathbf{a}_i$, $i\in I^+$, and the first $n^- - 1$ vectors $\mathbf{a}_i$, $i\in I^-\backslash\{k\}$, and assigning the remaining vector $\mathbf{a}_k$ according to

$$\mathbf{a}_k=\frac{\left(\sum_{i\in I^+}\lambda_i\mathbf{a}_i-\sum_{i\in I^-\backslash\{k\}}\mu_i\mathbf{a}_i\right)}{\mu_k},$$

where $I^-\backslash\{k\}$ denotes the set of indices I^- with k removed, and k is chosen so that $\mu_k\neq 0$.

III Choose a solution vector $\mathbf{b}^*$ and scalar $\zeta^*>0$.

IV Set the observations y_i, $i\in I^+\cup I^-$, using

$$y_i=\begin{cases}\mathbf{a}_i^{\mathrm{T}}\mathbf{b}^*-\zeta^*, & i\in I^+,\\ \mathbf{a}_i^{\mathrm{T}}\mathbf{b}^*+\zeta^*, & i\in I^-.\end{cases}$$

V Choose $m - n^+ - n^-$ vectors $\mathbf{a}_i$, $i\in I\backslash\{I^+\cup I^-\}$, and corresponding observations y_i, that define additional linear equations for which

$$\left|\mathbf{a}_i^{\mathrm{T}}\mathbf{b}^*-y_i\right|<\zeta^*,\qquad i\in I\backslash\{I^+\cup I^-\}.$$

VI Randomise the ordering of the vectors $\mathbf{a}_i$, re-ordering the observations y_i accordingly.

6.4 Continuity check on the square of complex number

In this section we illustrate how continuity checks (Section 5.2.2) are used to identify numerical properties of a simple calculation involving complex-valued quantities. The calculation is the square z^2 of complex z. The calculation may be performed in a number of ways, including

$$z^2=(z_R^2-z_I^2)+2iz_Rz_I, \tag{19}$$

where z is expressed in terms of its real and imaginary components,

$$z=z_R+iz_I,$$

and i is $\sqrt{-1}$, or

$$z^2=|z|^2e^{2i\arg(z)}, \tag{20}$$

where $|z|$ is the magnitude of z and arg(z) is its phase.

The advantage of (20) over (19) is that it can be generalised to non-integer powers. As a consequence, the calculation (20) may be used for computing z^2, whereas the calculation (19) is performed for the (mathematically equivalent) computation $z \times z$. (However, to avoid this some software packages and compilers will test whether the exponent is an integer, and apply (19) accordingly.)

Consider the range of complex numbers defined by $-1 \le z_R \le +1$ and $-10^{-100} \le z_I \le 10^{-100}$ that defines a narrow strip along the real axis. Figures 4 and 5 illustrate surface and contour plot representations of the function "imaginary part of z^2" calculated in the two ways. The function shown in Figure 4 which is calculated using (20) exhibits a sharp discontinuity along that part of the real axis corresponding to a negative imaginary part of z. In contrast, the function shown in Figure 5 which is based on (19) is, as expected, smooth.

Although the magnitude of the quantities is small, the non-smooth behaviour is undesirable, and may lead to difficulties when embedded within complex calculations. The behaviour is detected without the need for reference data sets. A comparison between the surfaces in the two figures may be used to quantify the errors in the given domain resulting from the use of (20).

The example (and, in particular, the range of z values chosen) is not pathological, it being the basis for a method using complex variables to estimate the derivative of real-valued functions [21]. The method is of great value in metrology applications in forming Jacobian matrices, sensitivity analysis, checking analytical derivatives, etc.

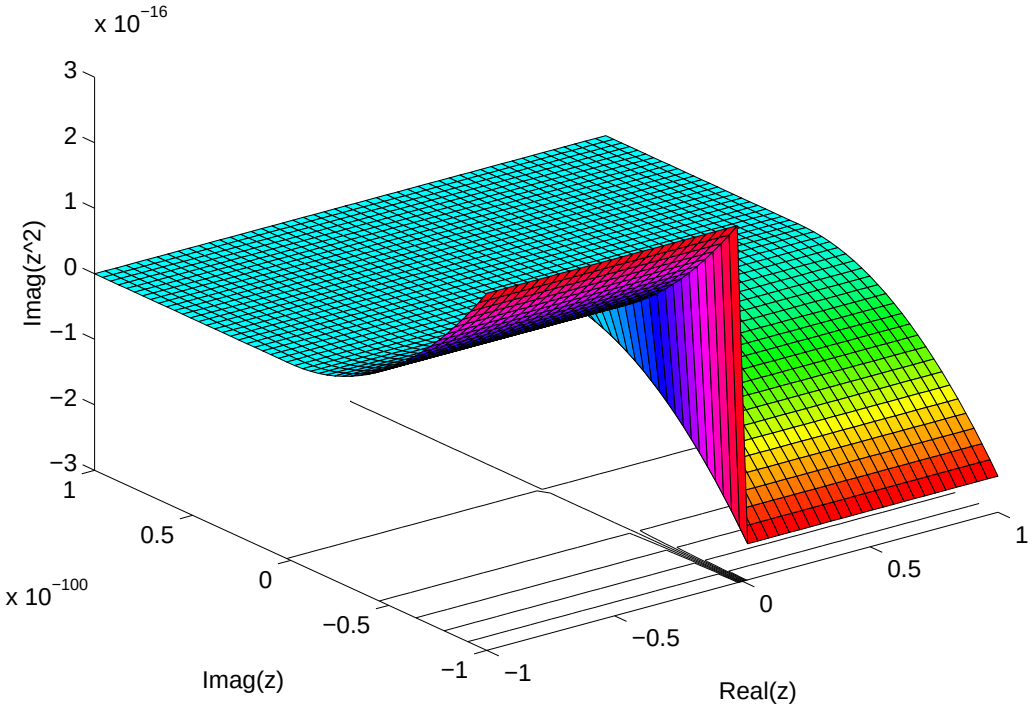

Figure 4 Values of the imaginary part of z^2 calculated using (20). The function exhibits a discontinuity.

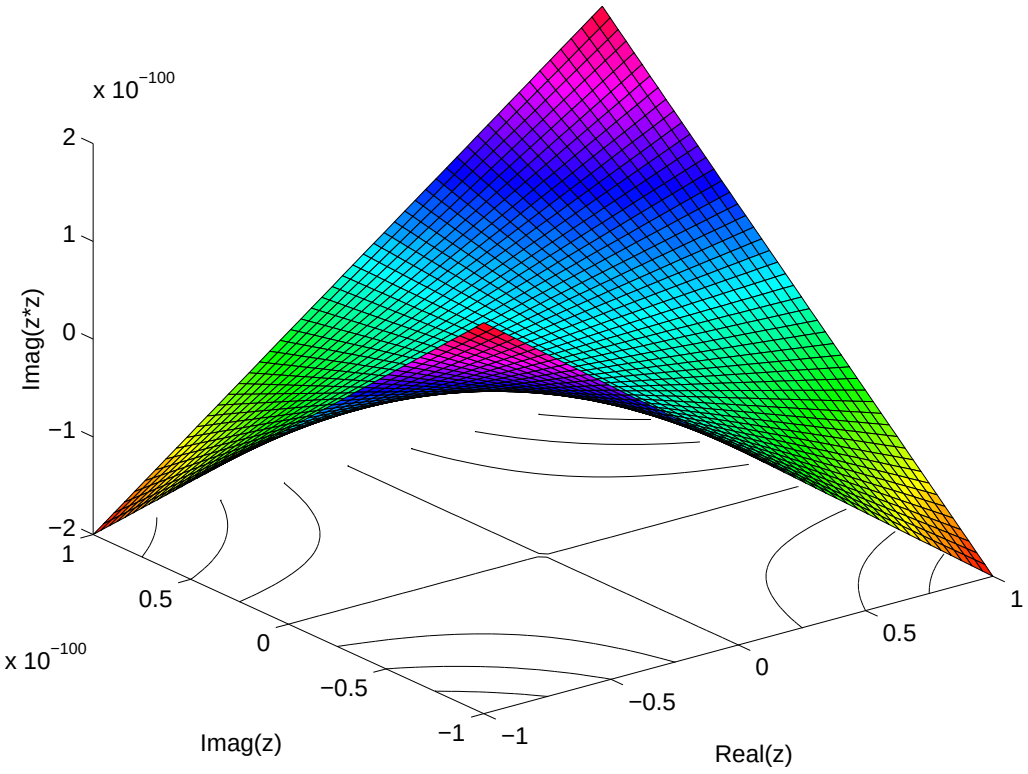

Figure 5 Values of the imaginary part of z^2 calculated using (19). The function is smooth, as expected.

7. Conclusions

In this report we have presented a general methodology for testing the numerical accuracy of scientific software. The basis of the approach is the design and use of reference data sets and corresponding reference results to undertake black-box testing. Reference data sets and results are generated in a manner consistent with the functional specification of the problem addressed by the software. In addition, data sets may be produced corresponding to problems with various "degrees of difficulty", or with application-specific properties. In these ways it is possible to make the testing representative of a particular metrology application, and to consider "fitness for purpose" of the test software. The results returned by the software for the reference data are compared objectively with the reference results. Quality metrics are used for this purpose that account for the key aspects of the problem: the absolute difference between the test and reference results, the computational precision used to generate the test results, and the condition of the problem.

The approach to testing comprises the following six stages:

1. specification of the test software,
2. implementation of the test software,
3. specification of reference data sets,
4. specification of performance measures and testing requirements,
5. generation of reference pairs, and
6. presentation and interpretation of performance measures.

The first two of these stages are usually carried out anyway as part of the software development process, although in practice with varying amounts of formality. Stages 3 to 6 constitute the approach to software testing advocated in this work. Each of these stages has been described,

and illustrated using various examples of scientific software used in metrology. Further examples are given in the companion reports [6, 7].

We have also described complementary tests that do not require the availability of reference results. These include consistency and continuity checks, spot checks against tabulated values, and checks of solution characterisations. The application of these tests can also be carried out in the spirit of the six stages indicated above. For example, for the consistency check

$$\sin^{-1}\{\sin x\} = x,$$

the variable x may be used as a performance parameter to define the range of inputs for which the test is to be applied, and the absolute difference between the values of $\sin^{-1}\{\sin x\}$ and x constitutes a performance measure for the check. Examples of the use of these complementary tests are provided in [7].

8. Acknowledgements

This report constitutes part of the deliverable of Project 2.1 of the 1998–2001 NMS Software Support for Metrology Programme, and has been funded by the National Measurement System Policy Unity of the UK Department of Trade and Industry.

9. References

[1] B.P. Butler, M.G. Cox, S.L.R. Ellison, and W.A Hardcastle, editors *Statistics Software Qualification: Reference Data Sets*. Royal Society of Chemistry, Cambridge, 1996.

[2] D.W. Lozier. A proposed software test service for special functions. In R.F. Boisvert, editor, *The Quality of Numerical Software: Assessment and Enhancement*, pages 167-178, London, 1997. Chapman and Hall.

[3] W. Van Snyder. Testing functions of one and two arguments. In R. F. Boisvert, editor, *The Quality of Numerical Software: Assessment and Enhancement*, pages 155-166, London, 1997. Chapman and Hall.

[4] M.G. Cox. Graded reference data sets and performance profiles for testing software used in metrology. In P. Ciarlini, M.G. Cox, F. Pavese, and D. Richter, editors, *Advanced Mathematical Tools in Metrology III*, pages 43-55, Singapore, 1997. World Scientific.

[5] ISO. ISO/DIS 10360-6. Geometrical product specifications (GPS) − acceptance test and reverification test for coordinate measuring machines (CMM). Part 6: Estimation of errors in computing Gaussian associated features, 1999. International Standards Organisation proposed draft standard.

[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*. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CISE 26/99, September 1999.

[7] H.R. Cook, M.G. Cox, M.P. Dainton and P.M. Harris. *Testing spreadsheets and other packages used in metrology: Testing the intrinsic functions of Excel*. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CISE 27/99, September 1999.

[8] G.T. Anthony, H.M. Anthony, B. Bittner, B.P. Butler, M.G. Cox, R. Drieschner, R. Elligsen, A.B. Forbes, H. Gross, S.A. Hannaby, P.M. Harris, and J. Kok. Reference software for finding Chebyshev best-fit geometric elements. *Precision Engineering*, 19:28-36, 1996.

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

[10] G.H. Golub and C.F. Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, MD, USA, 1996.Third edition.

[11] S.L.R. Ellison, M.G. Cox, A.B. Forbes, B.P. Butler, S.A. Hannaby, P.M. Harris and Susan M. Hodson. Development of data sets for the validation of analytical instrumentation. *J. AOAC International*, 77:777-781, 1994.

[12] T.F. Chan, G.H. Golub and R.J. LeVeque. Algorithms for computing the sample variance: analysis and recommendations. *The American Statistician*, **37**(3), 1983.

[13] T.F. Chan and J.G. Lewis. Computing standard deviations: accuracy. *Comm. ACM*, **22**(9), 1979.

[14] M. G. Cox and A. B. Forbes. Strategies for testing form assessment software. Technical Report DITC 211/92, National Physical Laboratory, Teddington, UK, 1992.

[15] B.P. Butler, M.G. Cox, A.B. Forbes, S.A. Hannaby, and P.M. Harris. A methodology for testing classes of approximation and optimisation software. In R. F. Boisvert, editor, *The Quality of Numerical Software: Assessment and Enhancement*, pages 138-151, London, 1997. Chapman and Hall.

[16] The MathWorks. *Matlab User's Manual*. MathWorks Inc., Natick, Mass., USA, 1992.

[17] J. N. Lyness. Performance profiles and software evaluation. In L. D. Fosdick, editor, *Performance Evaluation of Numerical Software*, pages 51-58, Amsterdam, 1979. North-Holland.

[18] G. T. Anthony and M.G. Cox. The design and validation of software for dimensional metrology. *National Physical Laboratory*, NPL Report DITC 50/84, 1984.

[19] P. Rees. Computing breathes new life into near infrared. *Scientific Computing World*, (38):18-19, 1998.

[20] R. G. Brereton. *Chemometrics: Applications of Mathematics and Statistics to Laboratory Systems*. Ellis Horwood, Chichester, England, 1990.

[21] W. Squire and G. Trapp. Using complex variables to estimate derivatives of real functions. *SIAM Rev*., 40: 110−112, 1998.

Appendix A Performance Measures for Software Testing

This appendix is concerned with some of the numerical-analysis issues that arise when carrying out the objective testing of mathematical software. Consider the model

$$\mathbf{y} = \mathbf{f}(\mathbf{x}),$$

where **x** is the vector of inputs, **f** is the vector-valued model and **y** is the vector of outputs. Here, **f** can be regarded as a *formula*, e.g., to calculate the standard deviation *y* of a set of values **x**. The "formula" may be complicated, e.g., it might be defined implicitly and represent, for example, the solution to a nonlinear least-squares problem.

We are concerned with testing the numerical correctness of software which "evaluates" the formula for values of the inputs that are regarded as admissible. Admissibility here relates to acceptability both from the viewpoint of the function and from the domain of applicability of the algorithm implemented by the software under test.

Let $\delta\mathbf{x}$ denote a small perturbation of **x** and $\delta\mathbf{y}$ the corresponding perturbation of **y**. The *relative condition number* κ of the model is then defined as

$$\kappa(\mathbf{x}) = \frac{\|\delta\mathbf{y}\|}{\|\mathbf{y}\|} \Big/ \frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|}. \tag{21}$$

It measures the relative change in the output relative to a relative change in the input.

Now,

$$\delta\mathbf{y} = \mathbf{f}(\mathbf{x} + \delta\mathbf{x}) - \mathbf{f}(\mathbf{x}) \approx J(\mathbf{x})\delta\mathbf{x},$$

where

$$J(\mathbf{x}) = \left[\frac{\partial f_i}{\partial x_j}\right]$$

is the Jacobian of $\mathbf{f}(\mathbf{x})$. Thus,

$$\kappa(\mathbf{x}) = \frac{\|J(\mathbf{x})\delta\mathbf{x}\|}{\|\mathbf{y}\|} \Big/ \frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \frac{\|J(\mathbf{x})\|\|\delta\mathbf{x}\|}{\|\mathbf{y}\|} \Big/ \frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} = \frac{\|J(\mathbf{x})\|\|\mathbf{x}\|}{\|\mathbf{y}\|}. \tag{22}$$

Let $\mathbf{y}^{(\mathrm{ref})}$ denote the reference output corresponding to inputs **x**. (This is the output that would be produced by applying reference software to the inputs. Such software is to be regarded as a sound implementation of an optimally-stable algorithm for evaluating the formula.) Furthermore, suppose $\mathbf{y}^{(\mathrm{test})}$ is the output produced by test software, and let

$$\Delta\mathbf{y} = \mathbf{y}^{(\mathrm{test})} - \mathbf{y}^{(\mathrm{ref})}.$$

We use (21) to "transform" $\|\Delta\mathbf{y}\|$ into an equivalent $\|\Delta\mathbf{x}\|$, in the sense that a $\Delta\mathbf{x}$ having this norm could give rise to this $\Delta\mathbf{y}$:

$$\|\Delta\mathbf{x}\| = \frac{\|\Delta\mathbf{y}\|}{\|\mathbf{y}^{(\mathrm{ref})}\|} \Big/ \frac{\kappa(\mathbf{x})}{\|\mathbf{x}\|} = \frac{\|\Delta\mathbf{y}\|}{\|\mathbf{y}^{(\mathrm{ref})}\|} \times \frac{\|\mathbf{x}\|}{\kappa(\mathbf{x})}. \tag{23}$$

For an optimally-stable algorithm, $\Delta\mathbf{y}$ would be such that the above corresponding $\Delta\mathbf{x}$ would be minimal, i.e., such that $\|\Delta\mathbf{x}\| \approx \eta\|\mathbf{x}\|$, where η is the relative machine precision. Such a $\Delta\mathbf{x}$ could arise from decimal-to-binary conversion on data input, for example. So, for high-quality or reference software, it can be expected that

$$\frac{\|\Delta \mathbf{x}\|}{\eta \|\mathbf{x}\|} = O(1), \tag{24}$$

or, equivalently, using (23),

$$\frac{1}{\kappa(\mathbf{x})\eta} \frac{\|\Delta \mathbf{y}\|}{\|\mathbf{y}^{(\mathrm{ref})}\|} = O(1). \tag{25}$$

For test software which is such that the left-hand side in (24) or (25) is $O(10^p)$, it can be concluded that for the inputs **x** it loses p more decimal figures than does the reference software. Therefore, the performance measure

$$P(\mathbf{x}) = \log_{10}\left(1 + \frac{1}{\kappa(\mathbf{x})\eta} \frac{\|\Delta \mathbf{y}\|}{\|\mathbf{y}^{(\mathrm{ref})}\|}\right) \tag{26}$$

estimates this value of p. It is important that this measure is applied to a sensibly-scaled problem.

The quantity (25) can be regarded as an error-amplification factor $EAF(\mathbf{x})$ for the test software. It can be written as

$$EAF(\mathbf{x}) = \frac{\rho(\mathbf{y}^{(\mathrm{test})})}{\kappa(\mathbf{x})\eta}, \tag{27}$$

where $\rho(\mathbf{y}^{(\mathrm{test})})$ is the relative error in $\mathbf{y}^{(\mathrm{test})}$,

$$\rho(\mathbf{y}^{(\mathrm{test})}) = \frac{\|\mathbf{y}^{(\mathrm{test})} - \mathbf{y}^{(\mathrm{ref})}\|}{\|\mathbf{y}^{(\mathrm{ref})}\|}.$$

An alternative representation of the quality metric is

$$P(\mathbf{x}) = \log_{10}(1 + EAF(\mathbf{x})),$$

where $EAF(\mathbf{x})$ is as in (27), or, using (22),

$$EAF(\mathbf{x}) = \frac{\|\Delta \mathbf{y}\|}{\|J(\mathbf{x})\|\|\mathbf{x}\|\eta}.$$

A.1 Example 1: Difference calculation

Consider the simple formula

$$y = x_1 - x_2. \tag{28}$$

Now,

$$J = \begin{bmatrix} 1 & -1 \end{bmatrix},$$

and, using (22),

$$\kappa = \frac{2^{1/2}(x_1^2 + x_2^2)^{1/2}}{|x_1 - x_2|},$$

or, using 1-norms,

$$\kappa = \frac{|x_1| + |x_2|}{|x_1 - x_2|}.$$

The latter result is identical to that from a specific analysis of the difference formula (28).

A.2 Example 2: Sample standard deviation

The mean $\bar{x}$ and standard deviation s of a set of values $\mathbf{x} = \{x_i: i = 1, \ldots, m\}$ are defined by

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

and

$$(m-1)s^2 = \sum_{i=1}^{m} (x_i - \bar{x})^2.$$

Now,

$$m\frac{\partial \bar{x}}{\partial x_i} = 1$$

and

$$(m-1)s\frac{\partial s}{\partial x_i} = (x_i - \bar{x})\left(1 - \frac{\partial \bar{x}}{\partial x_i}\right),$$

giving

$$\frac{\partial s}{\partial x_i} = \frac{x_i - \bar{x}}{ms}.$$

So

$$J = \frac{1}{ms}\left[(x_1 - \bar{x}), \ldots, (x_m - \bar{x})\right]$$

and

$$\|J\| = \frac{1}{ms}\left(\sum_{i=1}^{m} (x_i - \bar{x})^2\right)^{1/2} = \frac{(m-1)^{1/2}}{m}.$$

Thus, using (22),

$$\kappa(\mathbf{x}) = \frac{(m-1)^{1/2}}{m} \times \frac{\left(\sum_{i=1}^{m} x_i^2\right)^{1/2}}{s}.$$

But, since

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

we have

$$\kappa^2(\mathbf{x}) = \frac{m-1}{m^2 s^2}\left\{(m-1)s^2 + m\bar{x}^2\right\} = \left(\frac{m-1}{m}\right)^2 + \frac{m-1}{m}\left(\frac{\bar{x}}{s}\right)^2. \qquad (29)$$

The first term, $\{(m - 1)/m\}^2$, in (29) is a constant, depending only on the sample size, and is therefore irreducible. The second term, $\{(m - 1)/m\}(\bar{x}/s)^2$, is smallest when $\bar{x} = 0$. Since the standard deviation of a sample is invariant with respect to adjustment by a constant, correction by the mean $\bar{x}$ is desirable, since it renders the problem relative condition number as small as possible.

Finally,

$$EAF(\mathbf{x}) = \frac{\Delta s}{s\eta} \times \frac{1}{\left\{ \left(\frac{m-1}{m} \right)^2 + \frac{m-1}{m} \left(\frac{\bar{x}}{s} \right)^2 \right\}^{1/2}}.$$

A.3 Example 3: Least-squares regression

Let A denote an $m \times n$ observation matrix with $m \geq n$, $\mathbf{y}$ an observation vector with m elements and $\mathbf{b}$ a regression vector. The formal solution of the least-squares *linear* regression problem

$$\min_{\mathbf{b}} \|\mathbf{y} - A\mathbf{b}\|_2^2$$

is

$$\mathbf{b} = A^+\mathbf{y},$$

where A^+ denotes the pseudoinverse of A. The residuals $\mathbf{r}$ are given by

$$\mathbf{r} = \mathbf{y} - A\mathbf{b} = (I - AA^+)\mathbf{y},$$

where I is the identity matrix.

Suppose it is required to study the extent to which test software can correctly determine these residuals. Now,

$$J = \frac{\partial \mathbf{r}}{\partial \mathbf{y}} = I - AA^+,$$

which is a projection matrix, and so

$$\|J\| = 1.$$

Moreover, this result is independent of the parametrisation used. Thus, applying (22),

$$\kappa(\mathbf{y}) = \frac{\|\mathbf{y}\|}{\|\mathbf{r}\|},$$

and the corresponding performance measure is

$$P(\mathbf{y}) = \log_{10}\left(1 + \frac{\|\Delta \mathbf{r}\|}{\|\mathbf{y}\|\eta} \right).$$

We note that $\kappa(\mathbf{y})$ is (a measure of) the signal-to-noise ratio for the data.

The performance measure can be shown also to be applicable to the least-squares *nonlinear* regression problem

$$\min_{\mathbf{b}} \|\mathbf{y} - \mathbf{f}(\mathbf{b})\|_2^2,$$

where $\mathbf{f}(\mathbf{b})$ is a nonlinear function of the regression parameters $\mathbf{b}$.

Appendix B Reference Values of Less than Ideal Accuracy

Ideally, reference outputs would be available for purposes of software testing which possessed the maximum accuracy available, i.e., the number of correct figures would be that of a full floating-point word less the number of figures that corresponds to the almost inevitable loss of accuracy as a result of the condition of the problem underlying the data. Such a reference output is used within software performance measures, a vital constituent of which is the relative error in the output defined by

$$\frac{\left\|\boldsymbol{y}^{(\text{test})}-\boldsymbol{y}^{(\text{ref})}\right\|}{\left\|\boldsymbol{y}^{(\text{ref})}\right\|}. \tag{30}$$

In (30), $\boldsymbol{y}^{(\text{ref})}$ denotes the reference output and $\boldsymbol{y}^{(\text{test})}$ the test output, i.e., the results produced by the software under test.

For some problems, however, it is difficult to provide such high-quality reference outputs. Nevertheless, it may be more straightforward to provide approximations which are not of this demanding accuracy, but which can be expected to be moderately more accurate than the test outputs. Consideration is therefore given to using such an approximation, $\tilde{\boldsymbol{y}}$, say, in place of $\boldsymbol{y}^{(\text{ref})}$ in (30). By making plausible assumptions about the approximation, it will be shown that the quantity (30) is inflated by no more than 23%.

In order to show how an approximate reference value can be used to establish the above result, the following assumptions (other assumptions are of course possible, and inflation factors can be determined accordingly) are first made concerning a known approximation $\tilde{\mathbf{y}}$ to $\boldsymbol{y}^{(\text{ref})}$:

1. $\tilde{\boldsymbol{y}}$ is at least ten times more accurate than $\boldsymbol{y}^{(\text{test})}$.
2. $\tilde{\boldsymbol{y}}$ has at least one correct decimal figure of accuracy.

From Assumption 1,

$$\left\|\tilde{\mathbf{y}}-\mathbf{y}^{(\text{ref})}\right\| \le \frac{1}{10}\left\|\mathbf{y}^{(\text{test})}-\mathbf{y}^{(\text{ref})}\right\|, \tag{31}$$

and from Assumption 2,

$$\left\|\tilde{\mathbf{y}}-\mathbf{y}^{(\text{ref})}\right\| \le \frac{1}{10}\left\|\mathbf{y}^{(\text{ref})}\right\|. \tag{32}$$

Since

$$\left\|\mathbf{y}^{(\text{test})}-\mathbf{y}^{(\text{ref})}\right\| = \left\|(\mathbf{y}^{(\text{test})}-\tilde{\mathbf{y}})+(\tilde{\mathbf{y}}-\mathbf{y}^{(\text{ref})})\right\|,$$

it follows, using (31), that

$$\left\|\mathbf{y}^{(\text{test})}-\mathbf{y}^{(\text{ref})}\right\| \le \left\|\mathbf{y}^{(\text{test})}-\tilde{\mathbf{y}}\right\| + \frac{1}{10}\left\|\mathbf{y}^{(\text{test})}-\mathbf{y}^{(\text{ref})}\right\|,$$

giving

$$\left\|\mathbf{y}^{(\text{test})}-\mathbf{y}^{(\text{ref})}\right\| \le \frac{10}{9}\left\|\mathbf{y}^{(\text{test})}-\tilde{\mathbf{y}}\right\|. \tag{33}$$

Also, since

$$\left\|\tilde{\mathbf{y}}\right\| = \left\|\mathbf{y}^{(\text{ref})}+(\tilde{\mathbf{y}}-\mathbf{y}^{(\text{ref})})\right\| \le \left\|\mathbf{y}^{(\text{ref})}\right\| + \left\|\tilde{\mathbf{y}}-\mathbf{y}^{(\text{ref})}\right\|,$$

it follows, using (32), that

$$\|\tilde{\mathbf{y}}\| \le \left\|\mathbf{y}^{(\mathrm{ref})}\right\| + \frac{1}{10}\left\|\mathbf{y}^{(\mathrm{ref})}\right\|,$$

i.e.,

$$\left\|\mathbf{y}^{(\mathrm{ref})}\right\| \ge \frac{10}{11}\|\tilde{\mathbf{y}}\|. \tag{34}$$

From (33) and (34),

$$\frac{\left\|\mathbf{y}^{(\mathrm{test})} - \mathbf{y}^{(\mathrm{ref})}\right\|}{\left\|\mathbf{y}^{(\mathrm{ref})}\right\|} \le \frac{11}{9}\frac{\left\|\mathbf{y}^{(\mathrm{test})} - \tilde{\mathbf{y}}\right\|}{\|\tilde{\mathbf{y}}\|}. \tag{35}$$

Inequality (35) is the required result. It implies that $\tilde{\boldsymbol{y}}$ can be used in place of $\boldsymbol{y}^{(\mathrm{ref})}$ in the expression (30) for the relative error with an increase in the best bound for (30) of at most 23%. It is emphasised that the conditions under which (35) is applicable must hold, i.e., assumptions 1 and 2 must be appropriate. A different result would apply under different assumptions.