NPL Report CMSC 04/00

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

GUIDELINES TO HELP USERS SELECT AND USE SOFTWARE FOR THEIR METROLOGY APPLICATIONS

BY

M G COX and P M HARRIS

September 2000

Guidelines to help users select and use software for their metrology applications

M G Cox and P M Harris
Centre for Mathematics and Scientific Computing

September 2000

ABSTRACT

In this report we provide information and guidelines to help users select and use software in such a way that is fit for purpose for their metrology applications. Fitness for purpose is defined here in terms of meeting requirements relating to the numerical accuracy of the results returned by the software. Four topics are covered. Firstly, a discussion is given of some of the factors that affect how users decide fitness for purpose requirements, accounting for the possible ways the results returned by scientific software may be subsequently used. Secondly, approaches are presented for verifying or testing the numerical accuracy of scientific software. Thirdly, guidance is given on how to make better use of existing software by pre-processing the input data to such software and using appropriate model parametrisations. Finally, information is provided to help users locate software that is relevant to metrology applications.

This report constitutes one of the deliverables 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.

© Crown Copyright 2000
Reproduced by permission of the controller of HMSO

ISSN 1471–0005

Extracts from this report may be reproduced provided the source is acknowledged
and the extract is not taken out of context.

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

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

Contents

1. Introduction

The aim of this report is to provide information and guidelines to help users select and use software in such a way that is *fit for purpose* for their metrology applications in terms of meeting requirements relating to the *numerical accuracy* of the results returned by the software. Although there are other issues that we may consider to decide fitness for purpose (some are listed below), we focus on numerical accuracy because it directly impacts on the *quality* of measurements results.

The following topics are covered.

1. Fitness for purpose of software (in respect of the numerical accuracy it delivers).
2. Verifying existing software.
3. Making better use of existing software.
4. Locating relevant software.

Specifically, in respect of these topics:

a) How does a user decide whether software is fit for purpose within the context of a metrology application?

   Some possible interpretations of the term "fit for purpose" are presented in Section 2.

b) How does a user go about verifying that software undertaking a specific computational task meets (i) any claims made about the accuracy of the results returned by the software, and/or (ii) the requirements of the user in this regard?

   Methodologies for testing scientific software, including advice on the use of quality metrics to quantify fitness for purpose with guidance on how to interpret these quality metrics, are discussed in Section 3.

c) How might a user go about ensuring the reliability of the results returned by existing software for a specific computational task?

   Advice on the use of data pre-processing, and the choice of problem parametrisation, that is relevant to a range of mathematical and statistical procedures that are commonly used in metrology is given in Section 4.

d) Where might a user find scientific software that is relevant to metrology?

   A number of sources of such software are listed in Section 5.

In addition to the numerical performance of scientific software, some other issues that should be considered when selecting software tools include:

- performance[1],
- usability,
- training requirements,
- operational environment,
- data format compatibility,
- interoperability,
- maturity[2],

[1] For example, its computational performance in terms of memory requirements and execution time.

- supplier support, and
- ISO 9001 (and TickIT) registered supplier support.

These issues are not considered in this report.

The emphasis is on providing information and guidelines to help users make the best use of existing software that is presented to the user as a "black-box": hence, we concentrate on black-box testing, and what the user can do "externally" to influence the robustness and reliability of the software, for example, in terms of the way the input data is presented to the software and the manner in which the software is used. However, we emphasise that there is *no* substitute for using software that implements *provably* robust and reliable numerical algorithms based on good *numerical analysis* and supported by an *error analysis*, and the best advice for the user is to choose where possible such software, some examples of which are indicated in Section 5.

2. Fitness for Purpose of Software Used in a Metrology Context

The concern here is fitness for purpose of software in respect of the numerical accuracy it delivers for input data that is representative of an application domain. Before it can be decided whether an item of software is fit for purpose within the context of a metrology application, it is necessary to appreciate the possible interpretations of the term "fit for purpose", and the circumstances in which these interpretations are appropriate.

We begin by describing two measures of numerical accuracy, and indicate the circumstances in which these measures can be sensibly used (Section 2.1). We then describe situations in which the requirements placed on software described in terms of such measures need to be modified to account for the way the results returned by the software are subsequently used. These situations include those where the results returned by the software

1. are used as inputs to subsequent user computations (Section 2.2),
2. are used as inputs to interfaced software components (Section 2.3), and
3. may be correlated (Section 2.4).

2.1 Assessing numerical accuracy

Let x be the exact value of a scalar quantity, and y its computed value. There are two natural ways to assess the numerical accuracy of y. The first is its *absolute error*, defined by

$$|x - y|,$$

and the second is its *relative error*, defined by

$$\frac{|x - y|}{|x|}.$$

The measure of error that is most appropriate or "sensible" to use will depend on circumstances. Consider the following examples:

1. $x = 1$, $y = x + 1$ with absolute error 1 and relative error 1;
2. $x = 10^8$, $y = x + 1$ with absolute error 1 and relative error 10^{-8};
3. $x = 10^{-8}$, $y = x + 10^{-9}$ with absolute error 10^{-9} and relative error 0.1.

In the first case y would probably be regarded as a *poor* approximation to x and the absolute and relative errors are not small.

---

[2] That is, its integrity is supported by wide use over a long period and in a range of contexts.

In the second case, the absolute error is the same as for the first case, although y would probably be regarded as a *good* approximation to x because the difference between x and y is small in relation to the size of x: this is reflected in its relative error.

The third case illustrates the problems of using relative error when the exact value becomes close to zero. Here, y would probably be regarded as a *good* approximation to x: this is reflected in its absolute error.

An alternative measure [1], which combines the features of absolute and relative error, is

$$\frac{|x-y|}{1+|x|}.$$

This measure is such that it is similar to relative error when $|x| >> 1$, and to absolute error when $|x| << 1$.

By replacing "absolute values" in the above expressions by "lengths of vectors" (or vector norms) the scalar measures given above can be generalised to measure the numerical accuracy of vector quantities $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. Furthermore, the absolute and relative measures described here form the basis of the quality metrics and performance measures used for *assessing* the numerical performance of software: see Section 3.4.

2.2 Results that will be used as inputs to subsequent user computations

There are many instances in metrology where the results of one computation are required to feed into subsequent computations. For example, the calibration of a measurement system requires finding a model for the system by the solution of algebraic or differential equations, and may be followed by

- the evaluation of the model to predict the behaviour of the system for subsequent measurements, and
- the differentiation of the model to provide sensitivity coefficients for constructing an uncertainty budget.

In many cases, the computations are performed by different people within the measurement chain. Furthermore, the outputs of the first computation are frequently communicated by a document, such as a calibration certificate, which necessitates some loss of accuracy when presenting results. In order to decide the accuracy requirements for the output of the first computation, it is important to consider the requirements of subsequent computations.

As an example, consider the following (quadratic) least-squares regression problem, which arises, for example, when establishing calibration curves. We wish to determine values of the parameters a_0, a_1 and a_2 in the model

$$y = a_0 + a_1 x + a_2 x^2,$$

that minimise the function

$$\sum_{i=1}^{m} \left\{ y_i - \left(a_0 + a_1 x_i + a_2 x_i^2 \right) \right\}^2,$$

given data (x_i, y_i), $i = 1, ..., 11$, where the standard uncertainty of each y-value is 0.1, and the x_i are chosen to be equispaced between 100 and 101. For the particular data listed in Table 1, the solution obtained from an algorithm that implements a numerically-stable approach (based on matrix factorisations) to solve this problem, is

$$a_0 = 6302.694405559,$$
$$a_1 = -129.4314685308,$$
$$a_2 = 0.6643356643321,$$

where each value is quoted to thirteen significant figures. Now we may believe that it is sufficient to compute each parameter value to an (absolute) accuracy of 10^{-4}, or to four digits after the decimal point, and to present such rounded values on a calibration certificate as follows:

$$\hat{a}_0 = 6302.6944,$$
$$\hat{a}_1 = -129.4315,$$
$$\hat{a}_2 = 0.6643.$$

However, for the purposes of evaluating the model, we find that for x within the range of the data

$$\left|\left(a_0 + a_1 x + a_2 x^2\right) - \left(\hat{a}_0 + \hat{a}_1 x + \hat{a}_2 x^2\right)\right| \approx 0.4,$$

and, consequently, the differences between the model values obtained using the two sets of parameter values exceeds the measurement uncertainty in the original data.

Much of the difficulty in this example is caused by the particular *model parametrisation* that we have chosen to use to communicate the results of the regression. If the requirement is to be able to *evaluate* the model accurately, rather than to provide estimates of the parameters a_0, a_1 and a_2, then we shall see in Section 4.2 that there are better ways of communicating the necessary information.

i	x_i	y_i
1	100.0	3.0
2	100.1	3.1
3	100.2	3.7
4	100.3	4.0
5	100.4	4.2
6	100.5	4.9
7	100.6	5.3
8	100.7	5.6
9	100.8	6.1
10	100.9	6.5
11	101.0	7.0

Table 1: Measurement data for (quadratic) least-squares regression problem.

2.3 Results that will be used as inputs to interfaced software components

The need to provide optimal solutions that minimise a *cost function* is commonplace in metrology, in such areas as

- fitting physical and empirical models to measurement data, and
- experimental design.

An example in the former area is finding the best-fit sphere to measured coordinate data where we wish to determine the centre coordinates (x_0, y_0, z_0) and radius r of the sphere that minimise the function

$$\sum_{i=1}^{m} d_i^2,$$

where d_i is the radial deviation from the ith measurement point (x_i, y_i, z_i) to the sphere,

$$d_i = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2} - r, \quad i = 1,\ldots m.$$

An example in the latter area is the problem of determining where to take measurements in order to achieve an "accurate" calibration of an artefact. For example, deciding where to take measurements on a sphere so that the standard uncertainty of the radius r of the solution to the least-squares sphere fitting problem is minimised.

The cost function is usually expressed as a mathematical function that is evaluated using software, and is provided as input to, or is interfaced to, general-purpose optimisation software. If the software for evaluating the function returns results to only modest accuracy, the performance of the optimisation software can be compromised and, in particular, convergence to the solution can be strongly affected.

For example, consider the minimisation of the function $f(x)$. In the neighbourhood of the minimum $x = x_0$, $f(x)$ can be expressed as

$$f(x) = f(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0),$$

assuming it is sufficiently smooth, since $f'(x_0) = 0$ at the minimum. It follows that, for x near x_0,

$$x - x_0 = \sqrt{\frac{2(f(x) - f(x_0))}{f''(x_0)}}.$$

Suppose that $f(x)$ is evaluated using software that returns results to an absolute accuracy of order ε, and that the scale of x is such that $f''(x_0)$ is of order unity. Then, since $f(x) - f(x_0)$ is of order ε, it follows that $x - x_0$ is of order $\sqrt{\varepsilon}$. In other words, the number of digits that are correct in determining the solution x_0 is approximately half that returned by the software for evaluating $f(x)$. Consequently, the accuracy requirements placed on the software for evaluating $f(x)$ need to be more stringent than those for the required solution, and it is typically the latter that are specified in user requirements.

2.4 Results that may be correlated

It is widely accepted that the use of two significant figures is adequate in quoting the uncertainties of measurement results. In many circumstances this is indeed the case. There are, however, extremely important exceptions to this rule. They arise when providing measurement results that are (likely to be) correlated. As an example, consider two measurement results whose errors are correlated. Rather than stating just the uncertainties (or variances) associated with the two results, the covariance would also be quoted. Suppose that the variances are 0.367 and 0.200 (rounded to three significant figures, in some appropriate units) and that the covariance is –0.270. The result would probably be recorded as the covariance matrix

$$V = \begin{bmatrix} 0.367 & -0.270 \\ -0.270 & 0.200 \end{bmatrix}.$$

Suppose, however, that in a particular application, the correlation matrix P was required. P is easily computed from V and, to three significant figures, is

$$P = \begin{bmatrix} 1.00 & -0.997 \\ -0.997 & 1.00 \end{bmatrix}.$$

Recording this result to *two* significant figures yields

$$P = \begin{bmatrix} 1.0 & -1.0 \\ -1.0 & 1.0 \end{bmatrix},$$

which, in subsequent use, could well be given the interpretation that the errors in the measurement results are *perfectly (negatively) correlated*. This conclusion, which would appear reasonable, given *only* the *P*-matrix immediately above, is of course, false.

Matrices having properties similar to those of *V* above arise commonly. For this example, *V* is in fact the covariance matrix for the parameters representing the intercept and the gradient of a straight-line least-squares regression model for the data points (x_i, y_i), $i = 1, 2, 3, 4,$ where the standard uncertainty of each *y*-value is 0.1, and the x_i have the (relatively exact) values 1.2, 1.3, 1.4 and 1.5. (Note that the high correlation between the intercept and gradient parameters in this example results from the particular choice of x_i data, and may be reduced by centring the x_i data about its mean value (Section 4.1) or, equivalently, making use of a better parametrisation of the straight-line model (Section 4.2).)

3. Testing Scientific Software

We give an overview (see [2–5]) of two approaches to testing scientific software, viz., based on using reference software (Section 3.1) and data generators (Section 3.2) to undertake "black-box" testing. If such approaches are to be used in the context of establishing fitness for purpose for an application domain we require two further ingredients: firstly, to specify and generate data inputs that are representative of the application domain; secondly, to specify measures or quality metrics to express the claims made about, or requirements of, the test software and to verify such claims or requirements. These two aspects are described, respectively, in Sections 3.3 and 3.4.

It is important to realise that no testing approach is infallible. The approaches described below both use software to generate data and corresponding results, and consequently depend on the correctness of that software. However, the intention is to make the testing objective and to base it on sound principles: in particular, this is reflected in the use of data generators to produce reference data and corresponding reference results, and quality metrics to assess the performance of test software on such data.

3.1 Testing using reference software

The procedure for testing using reference software is illustrated in Figure 1. Starting with a reference data set, *reference software* and the *test software* are each applied to the reference data set to produce, respectively, corresponding reference results and test results. The reference and test results are then compared using quality metrics (Section 3.4).

The approach 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 for the test software. 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 [2], however, that reference software is very demanding to provide.

An example of reference software in the area of dimensional metrology is described in [6] for computing Chebyshev best-fit geometric elements to coordinate data. Fourteen items of reference software were developed in a 10.5 man-year project (an average of nine man-months per software item) undertaken by the National Physical Laboratory (NPL, UK), Physikalisch-

Technische Bundesanstalt (PTB, Germany) and Centrum voor Wiskunde en Informatica (CWI, Netherlands). Examples of reference software for more general scientific computations are listed in Section 5.

Of course, reference software does not have to be "best-possible": it is necessary only to use software that is "sufficiently better" than that required by the user, i.e., software provided with a (verified) claim about its accuracy that is better than that required by the user. Again, making the comparison with a measurement traceability chain for physical standards, we do not have to calibrate everything against a primary standard; instead it is sufficient to choose a standard that is higher in the measurement hierarchy provided that its accuracy is established (so the accuracy claim can be verified) by traceability to a primary standard. Furthermore, it is shown in [3, Appendix B] that using a reference result that is not best-possible has only a mild influence on the performance measure $P(\mathbf{x})$ described in Section 3.4.3 for assessing the numerical accuracy of test software.

One approach to generating high accuracy reference results is to undertake calculations using multiple or extended precision. This is the approach used by the National Institute of Standards and Technology (NIST, US) to provide data sets and certified results through its Statistical Reference Datasets (StRD) web-site [7]. Data is read in as multiple precision numbers and all calculations are made to a very high precision (accurate to 500 digits). The results are also output in multiple precision, and only then rounded to fifteen significant digits, which is the computational precision to which the majority of users will be working. The results represent what would be achieved if calculations were to be made without round-off or other errors. Any typical numerical algorithm that is not implemented using multiple precision arithmetic will introduce computational errors, and the comparison of test and reference results needs to account for this. Consequently, even though in this approach the reference results are not generated using reference software, if the comparison of test and reference results is to be meaningful it is essential to provide information on the accuracy to which reference software working to fifteen significant digits would return results. This is the motivation behind the performance measure $P(\mathbf{x})$ described in Section 3.4.3. Using NIST's data sets and certified results with this performance measure would provide a powerful approach to testing scientific software.

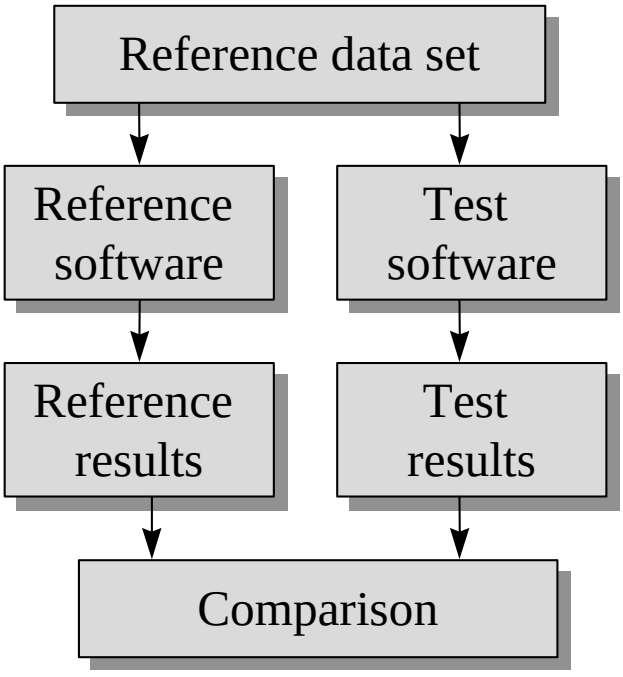

Figure 1 Procedure for testing software using reference software.

3.2 Testing using a data generator

The procedure for testing using a data generator is illustrated in Figure 2. Starting with some reference results, a *data generator* is used to produce a corresponding reference data set. The *test software* is applied to the reference data set to produce corresponding test results that are then compared with the reference results using quality metrics (Section 3.4).

The approach uses a data generator to construct reference data sets to have known solutions (constituting reference results) that are 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 problems solved in metrology using software. 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 determining reference results from reference data sets specified *a priori*.

For a problem 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 that mimic actual data sets from applications (Section 3.3). 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.

Examples of data generators are given in [4] for the linear least-squares regression problem

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

and the linear minimax regression problem

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

in [5] for fitting a Gaussian peak to measurement data, in [8] for fitting geometric elements to coordinate measurement data, and in [9] for the calculation of effective area in the calibration of pressure balances.

3.3 Specifying representative data sets for an application domain

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

1. 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 "similar to" the data sets used in a particular application domain,
2. the construction of data sets with known properties, e.g., the condition or "degree of difficulty" of the corresponding problem, and
3. the need to generate sufficient numbers of reference data sets to ensure adequate coverage of the space of possible inputs to the test software.

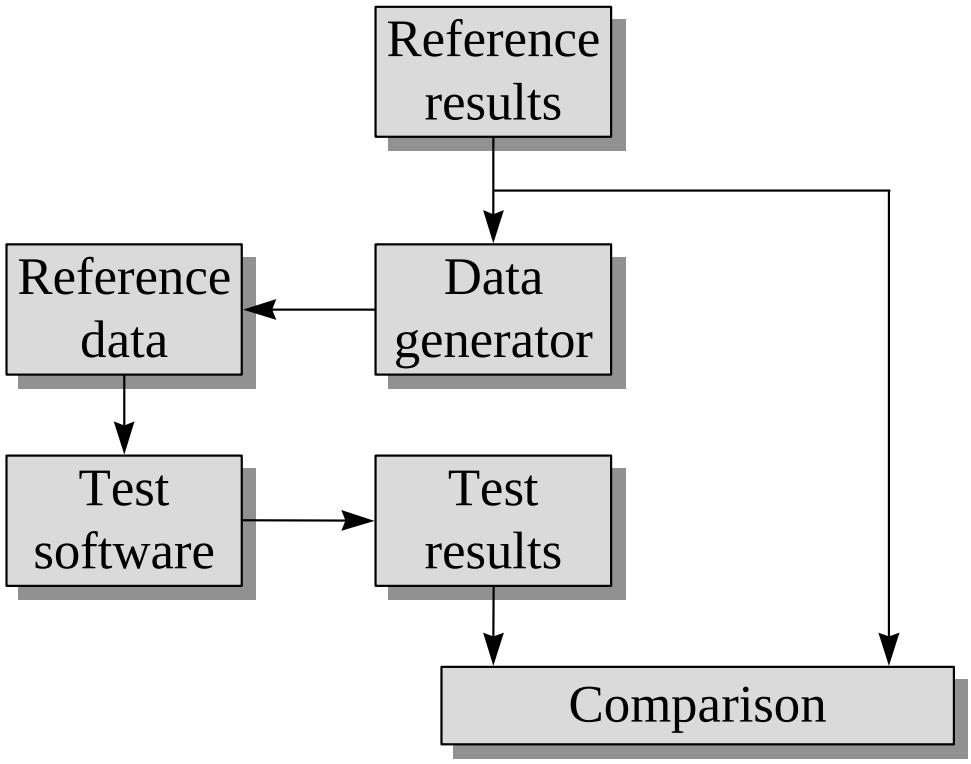

Figure 2 Procedure for testing software using data generators.

In many cases it is possible to "parametrise" the space of possible data inputs using (so-called) *performance parameters*. Then, *sequences* of data sets are generated for which, within a given sequence, different data sets correspond to different choices of a specified performance parameter (or parameters). 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 corresponding problem. 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.

By assigning suitable values or ranges of values to the performance parameters that specify the reference data sets, it is possible to control the data sets used in the testing so that they are representative of an application domain. 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 inherit (physical) properties: this aspect is described in [3, Section 4.1].

Example 1.

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.

The parameter $\bar{x}/s$, the inverse of the coefficient of variation expresses the condition of the problem of calculating the sample standard deviation of a data set [3, Section 1.2] and may be used to define a graded sequence of reference data sets.

Example 2.

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

a) describe 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 generated to inherit this property.

3.4 Specifying quality metrics and performance requirements

Whichever approach to testing is used, we must compare objectively the reference and test results. Furthermore, as we are concerned with fitness for purpose we must then compare the result of this comparison with

a) any *claim* about the software made by the *developer* (or *provider*) of the software, and/or

b) any *requirement* on the software made by the *user* of the software.

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, if the developer's claims and/or user's requirements are expressed in similar terms, we can use such metrics to assess objectively whether the test software is fit for purpose.

Example 1.

Let X denote a random variable having a Binomial distribution with parameters $n \geq 0$ and $0 < p < 1$, i.e.,

$$\text{Prob}(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \quad k = 0,\ldots,n.$$

Routine G01BJF from the NAG Fortran library [32] returns for given n, p and k, the probabilities Prob($X \leq k$), Prob($X > k$) and Prob($X = k$). The documentation for the routines states that results are correct to a relative accuracy of at least 10^{-6} on machines with a precision of nine or more decimal digits, and to a relative accuracy of at least 10^{-3} on machines of lower precision (provided that the results do not underflow to zero). This is a *claim* about the software made by the software developer.

Example 2.

The Microsoft Excel spreadsheet package contains a worksheet function TINV that returns the t-value of the Student's t-distribution for a given probability and number of degrees of freedom. The on-line help documentation provided by the software package states that "TINV iterates until the result is accurate to within $\pm 3 \times 10^{-7}$" and "if TINV does not converge after 100 iterations, the function returns the #N/A error value". This is a *claim* about the software made by the software developer. Note that there is some ambiguity in that it is not stated whether the accuracy of the results is measured in an absolute or relative sense.

Example 3.

The ISO Draft International Standard [10] is concerned with testing software for computing Gaussian best-fit geometric elements to coordinate measurement data used in industrial inspection and geometric tolerancing applications. The Standard requires that the parameters returned by test software are accurate to within prescribed absolute tolerances. For example, the

orientation of geometric elements such as the plane, cylinder and cone is defined by a unit direction vector, and the vector returned by test software is compared with a reference solution by comparing 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. This is a *requirement* on the accuracy of results returned by test software.

The quality metrics described below 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 (Section 3.4.1),
2. the number of figures of agreement, a *relative* measure of accuracy (Section 3.4.2), and
3. a *performance measure* that indicates the number of significant figures *unnecessarily* lost by the test software (Section 3.4.3).

The metrics are used in the following way. It is necessary to decide whether absolute or relative errors are appropriate. If absolute errors are important, the user should decide whether the value of $d(\mathbf{x})$, the absolute difference between the reference results and test results, is acceptably small. If relative errors are important, the user should decide whether $N(\mathbf{x})$, the number of figures of agreement between the reference results and the test results, is adequate for the application. The performance measure $P(\mathbf{x})$ indicates the number of figures of accuracy lost by the test software in computing the test results compared with reference software. It accounts for factors including the computational precision of the arithmetic used to generate the test and reference results and the condition of the problem. A large value of $P(\mathbf{x})$ may indicate that the algorithm employed by the test software is unstable. If, for a set of graded data sets taken in order, the performance measure $P(\mathbf{x})$ has a tendency to increase, it is likely that an unstable algorithm has been used.

3.4.1 Absolute measures of accuracy

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

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

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

$$d(\mathbf{x}) = RMS(\Delta\mathbf{y}) \tag{1}$$

is an absolute measure of departure of the test results from the reference results for the reference data $\mathbf{x}$. Here,

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

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

$$d(\mathbf{x}) = \left| y^{(\text{test})} - y^{(\text{ref})} \right|. \tag{2}$$

In cases where the components of $\mathbf{y}$ are not comparably scaled, some care is required when interpreting the summary measure $d(\mathbf{x})$ defined by (1), and consideration of the component-wise errors (2) is recommended.

3.4.2 Relative measures of accuracy

Let $d(\mathbf{x})$ be defined by (1) or (2). Then, the number of figures of agreement $N(\mathbf{x})$ between the test results and reference results corresponding to reference data $\mathbf{x}$ is calculated from:

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

$$N(\mathbf{x}) = \min\left\{ M(\mathbf{x}), \log_{10}\left(1 + \frac{RMS(\mathbf{y}^{(\mathrm{ref})})}{d(\mathbf{x})} \right) \right\}. \tag{3}$$

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 $\mathbf{x}$, 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 3.4.1, (3) may be applied to the complete vector $\mathbf{y}$ or its components.

3.4.3 Performance measures

In [3, Appendix A] the performance measure $P(\mathbf{x})$ is derived, where

$$P(\mathbf{x}) = \log_{10}\left(1 + \frac{1}{\kappa(\mathbf{x})\eta} \frac{\|\Delta\mathbf{y}\|}{\|\mathbf{y}^{(\mathrm{ref})}\|} \right), \tag{4}$$

corresponding to reference data $\mathbf{x}$. 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 $\mathbf{y}^{(\mathrm{test})}$ and $\mathbf{y}^{(\mathrm{ref})}$ 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 [11].

4. Advice on the Use of Scientific Software

In this section we discuss ways in which a user of existing software for a specific computational task can help to ensure the reliability of the results returned by that software. As in Section 3 we assume that the software is regarded as a “black-box” and, consequently, the user cannot interfere with the (internal) operations performed by the software. Instead the main option available to the user is to *pre-process* or *transform* the input data prior to application of the software (and perhaps followed by appropriate post-processing of the output result). Illustrative examples of data transformations are presented in Section 4.1.

Because of the role of measurement *models* in metrology, and the importance of being able to work effectively with such models, e.g., in calibration problems, we consider separately the issue of *model parametrisation* and its influence on the results returned by software operating with such models. Again the emphasis here is on ways of presenting the models and associated measurement data to existing software so that the reliability of the results is not compromised. Model parametrisation is considered in Section 4.2.

Although the advice provided in this section can help to ensure the reliability of the results returned by existing software, it is emphasised again that there is no substitute for using

software that implements *provably* robust and reliable numerical algorithms based on good *numerical analysis* and supported by an *error analysis*. The use of such software (if it is available) should always be the preferred option, with the advice presented here being used to prevent some of the more obvious numerical problems caused by less reliable software.

4.1 Data transformations

We illustrate using a number of examples how the use of pre-processing of input data and corresponding post-processing of output results can help to ensure the reliability of the results returned by existing software. The examples include:

- *sorting* the values in a sample prior to the computation of the sample mean,
- *shifting* or *centring* the values in a sample prior to the computation of the sample standard deviation,
- *scaling* the values in a sample prior to the computation of the sample standard deviation,
- the use of *mathematical identities*: calculating $\pm f(-x)$ rather than $f(x)$ for an even or odd function, respectively, in cases where a function has been implemented stably only for positive arguments (or for negative arguments), and
- *replacing* the computation of the geometric mean of the values in a sample by the exponential of the arithmetic mean of the logarithms of the sample.[3]

In all these examples it is important that the transformation should be chosen so that the problem solved is *not* changed. For example, taking logarithms of the values in a sample to calculate their geometric mean as in the last example can be used as means to undertake this calculation more reliably. However, some software [4, 16] for fitting an exponential model to measurement data will do so by taking logarithms of the data and applying a procedure for linear least-squares regression to the transformed data. The approach avoids having to solve a non-linear least-squares problem. Although the procedure for the linear regression may be more reliable than that for a non-linear regression, the solutions to the two problems are different, and numerical error is introduced in this way.

Example 1.

Sorting the values in a sample $\{x_i: i = 1, \ldots, m\}$ into increasing order of magnitude can be expected to improve the accuracy of the computed sample mean $\bar{x}$ if the values have wide variation. The reason is that, by combining small values first, it is less likely that a small value is added to a large value, and hence that some of the significant figures in the small value are lost. This result is fully supported by an error analysis.

Given software for computing the sample mean, the approach is summarised by the following steps:

1. Pre-process the input data by sorting the values in the sample into increasing order of magnitude.
2. Use the software to calculate the sample mean of the sorted values. The value obtained is taken as the mean of the original sample.

Example 2.

Some statistical procedures for computing the standard deviation s of a sample $\{x_i: i = 1, \ldots, m\}$ use the numerically unreliable (one-pass) formula

[3] This last example is slightly different from the others in that we transform the input data and apply a *different* numerical procedure. Nevertheless, as we shall see, it has the desired effect.

$$s = \sqrt{\frac{\sum_i x_i^2 - \frac{1}{m}\left(\sum_i x_i\right)^2}{m-1}}. \qquad (5)$$

The formula has the property that for data sets for which s is small compared with the sample mean $\bar{x}$ it suffers from subtractive cancellation when forming the quantity

$$\sum_i x_i^2 - \frac{1}{m}\left(\sum_i x_i\right)^2 = \sum_i x_i^2 - m\bar{x}^2.$$

To illustrate, consider the sample {1.00940, 1.00970, 1.01000, 1.01030, 1.01060}. Then, working to six significant figures, we have

$$\sum_{i=1}^{5} x_i^2 = 1.00940^2 + 1.00970^2 + 1.01000^2 + 1.01030^2 + 1.01060^2 = 5.10050,$$
$$m\bar{x}^2 = 5(1.01000)^2 = 5.10050,$$

and *all* six significant figures are lost in forming

$$\sum_{i=1}^{5} x_i^2 - m\bar{x}^2 = 0.00000.$$

The consequence of using (5) is a loss of accuracy, which is extreme in this example, in computing s. Such accuracy loss can be avoided by subtracting the sample mean $\bar{x}$ (or an approximation to it) from all the values in the sample before using the procedure. This process of *shifting* or *centring* the sample, also referred to as *coding*, has the property that the value of the sample standard deviation is mathematically unchanged by the shift. However, numerically the sample standard deviation of the shifted data can be determined more reliably, particularly if only low computational precision is available.

For the above sample, $\bar{x} = 1.01000$, and a shifted version of the sample is $10^{-4} \times$ {–6.00000, –3.00000, 0.00000, 3.00000, 6.0000}. Then, working to six significant figures, we have

$$\sum_{i=1}^{5} x_i^2 = 10^{-8} \times \left(6.00000^2 + 3.00000^2 + 0.00000^2 + 3.00000^2 + 6.00000^2\right) = 10^{-6} \times 0.900000,$$
$$m\bar{x}^2 = 5(0.00000)^2 = 0.00000,$$

and *no* significant figures are lost in forming[4]

$$\sum_{i=1}^{5} x_i^2 - m\bar{x}^2 = 10^{-6} \times 0.900000.$$

Notice that by replacing x_i with $x_i - \bar{x}$ in (5), we obtain

$$s = \sqrt{\frac{\sum_i (x_i - \bar{x})^2 - \frac{1}{m}\left(\sum_i (x_i - \bar{x})\right)^2}{m-1}} = \sqrt{\frac{\sum_i (x_i - \bar{x})^2}{m-1}},$$

(since $\sum_i (x_i - \bar{x}) = 0$ constitutes a definition of the mean $\bar{x}$), which is the usual (two-pass) formula for s and is known to be numerically reliable.

Given software for calculating the sample standard deviation, the approach is summarised by the following steps:

[4] This is an extreme example in one respect. However, examples very close in their nature to this arise regularly in metrology.

1. Calculate the sample mean.
2. Pre-process the input data by subtracting the sample mean from all the values in the sample.
3. Use the software to calculate the sample standard deviation of the shifted values. The value obtained is taken as the standard deviation of the original sample.

Shifting the sample can also help to prevent *overflow* occurring, and thus improve the *reliability* of a provided statistical procedure. Let R denote the overflow threshold, i.e., the largest positive storable number, and consider applying the formula (5) to the sample {R/10, R/10, R/10}. Then, although the individual data samples may be represented on the computer, their squares may not. Even though the standard deviation for such a sample may be represented in floating-point form (it is zero), a numerical procedure implementing the formula will be unable to compute the statistic because overflow occurs when an intermediate quantity, the sum of squares, is calculated. However, the standard deviation of the sample shifted by its mean is correctly computed by the procedure.

Example 3.

If the values in the sample are all very large or very small in magnitude, it can be advantageous to *scale* them before computing the sample standard deviation, and then "de-scale" the computed result. Such scaling virtually eliminates the possibility of computer overflow or underflow. Some statistical procedures do not guard against this possibility.

Given software for calculating the sample standard deviation, the approach is summarised by the following steps:

1. Determine the value in the sample that has largest magnitude.
2. Pre-process the input data by scaling all the values in the sample by this quantity. (The largest value of the processed data, as a result, is unity.)
3. Use the software to calculate the sample standard deviation of the scaled values.
4. Post-process the result by de-scaling the computed sample standard deviation by the original scaling quantity. The value obtained is taken as the standard deviation of the original sample.

Let r denote the underflow threshold, i.e., the smallest positive storable number. Then, the sample {$-r$, $+r$} will lead to underflow in the application of formula (5) because although the individual samples are representable on the computer, their squares are too small to be represented and are stored as zero. In this case, shifting the sample gives no benefit (because the mean of the sample is zero): however, applying the formula to the data after it has been scaled avoids the problem and allows the sample standard deviation to be calculated correctly.

There is a variant of the scaling procedure given above for avoiding overflow and underflow. It has the additional property that for problems for which there is *no* overflow or underflow the results with and without scaling are *identical*. In this variant, rather than choosing the value X in the sample of largest magnitude, the scaling factor is chosen to be the smallest power of two[5] that is no smaller than the magnitude of X.

Example 4.

The problem of evaluating the inverse hyperbolic sine function for a given x is to solve for y the equation

$$x = \frac{e^y - e^{-y}}{2}.$$

[5] Or the radix of the computer's floating point arithmetic.

Multiplying by e^y, we obtain

$$(e^y)^2 - 2xe^y - 1 = 0,$$

which is a quadratic equation in e^y. The roots of this equation are given *mathematically* by

$$e^y = x \pm \sqrt{1+x^2},$$

and the positive root, which determines the required solution y, is

$$e^y = x + \sqrt{1+x^2}. \tag{6}$$

Now, for positive x, e^y is evaluated as the sum of two positive terms. However, for negative x, e^y is evaluated as the difference of two positive terms whose magnitudes become larger and relatively closer as x becomes more negative. The evaluation of e^y for negative x is therefore subject to subtractive cancellation. Software for evaluating the inverse hyperbolic sine function that does so by evaluating (6) will return results that are accurate for positive x but that become increasingly less accurate as x becomes more negative.

However, the inverse hyperbolic sine function is mathematically an *odd* function, i.e.,

$$\sinh^{-1}(x) = -\sinh(-x),$$

and, consequently, values of the function for $x < 0$ may be calculated from values for $|x|$ that are obtained without the possibility of subtractive cancellation. There are many other instances where mathematical identities or properties may be used in this way.

Given software for computing the inverse hyperbolic sine function, the approach is summarised by the following steps:

1. If the input data value is zero or positive, use the software to calculate the value of the inverse hyperbolic sine function.
2. If the input data value is negative:
   2.1. Pre-process the input data value by forming the modulus of its value.
   2.2. Use the software to calculate the value of the inverse hyperbolic sine function for the processed value.
   2.3. Post-process the result by forming its negative value. The value obtained is taken as the value of the inverse hyperbolic function for the original input data value.

Example 5.

The geometric mean $\bar{x}_g$ of a sample $\{x_i: i = 1, \ldots, m\}$ is defined by

$$\bar{x}_g = \left(\prod_{i=1}^{m} x_i \right)^{1/m}. \tag{7}$$

Most software for computing the geometric mean will produce values for modest values of m that have a relative accuracy approaching the computational precision η. However, for large m the same software can fail because of overflow or produce zero because of underflow. For example, consider an instrument that records 1024 values (m = 1024) each of which is approximately 2 ($x_i \approx 2$). The direct formation of the product in (7) would yield a value of the order of $2^{1024} \approx 10^{308}$ which could cause overflow on many computers even though the geometric mean is of the order of 2. Similarly, if each value is approximately 0.5 ($x_i \approx 0.5$), formation of the product could underflow and the value zero rather than a value of the order of 0.5 would be returned.

Since the geometric mean is identical to the arithmetic mean in a transformed variable, i.e.,

$$\ln \bar{x}_g = \frac{1}{m}\sum_{i=1}^{m} \ln x_i, \tag{8}$$

the use of (8) provides an approach in which overflow and underflow are avoided and a value having a relative accuracy approaching η is returned (although the use of the "ln" and "exp" functions can introduce some small additional loss of accuracy).

Given software for computing the sample arithmetic mean, the approach is summarised by the following steps:

1. Pre-process the input data by forming the (natural) logarithms of the values in the sample.
2. Use the software to calculate the arithmetic mean of the transformed values.
3. Post-process the result by forming the inverse (natural) logarithm of the result. The value obtained is taken as the geometric mean of the original sample.

4.2 Model parametrisation

Model parametrisation can readily be illustrated using the example of a straight-line model. A straight line can be expressed in a variety of ways, including

$$y = a_0 + a_1 x,$$

where a_0 is the y-axis intercept and a_1 is the gradient, and

$$y = a_0 + a_1 (x - c),$$

where a_1 is the gradient as before, but a_0 now represents the intercept with the line $x = c$. The latter form is clearly a generalisation of the former (which corresponds to the choice $c = 0$). This generality can be used to advantage when designing, for example, an algorithm for straight-line fitting, by selecting a value of c that yields desirable properties. The choice of the arithmetic mean of the x_i data values for c has several such properties (see example 1 below).

As illustrated above, and in the examples following, the way a model is specified is generally not unique, and this freedom can be exploited to ensure the use of software is made as reliable as possible. Furthermore, in many metrology applications we *genuinely* have the freedom to choose a parametrisation to suit our purposes. For example, if all we are interested in are the values of the above straight-line model at certain points within the range of the data, then we can choose a parametrisation of the model to give us accurate results. However, there will be some applications where the model parametrisation is fixed, because there is a requirement to estimate a certain parameter with physical significance. Nevertheless, even for such applications we can consider the two stages of (a) employing a sound parametrisation in our computations, and (b) calculating the parameter(s) of interest from this. Whether (b) confers any advantage for the metrology problem over direct use of a restricted parametrisation then needs to be considered.

Example 1.

A polynomial $p(x)$ of degree n may be represented using the monomials x^j as the power series

$$p(x) = a_0 + a_1 x + \ldots + a_n x^n.$$

However, for $x > 0$, the basis functions 1, x, ..., x^n in this representation tend to be "similar" in appearance, and consequently there is numerical instability when used in polynomial regression. Typically, the coefficients a_j assume a wide range of numerical values, and considerable cancellation (and hence loss of figures) can take place when evaluating the polynomial expressed in this way.

The situation is improved if the independent variable x is normalised to the interval $[-1, +1]$. If x is in the range $[x_{\min}, x_{\max}]$, this is achieved by the transformation

$$X = \frac{(x - x_{\min}) - (x_{\max} - x)}{(x_{\max} - x_{\min})}.$$

There is low correlation between even and odd powers of X of low to moderate degree, and the power series representation in terms of X may be safe in such circumstances.

Recall the quadratic polynomial regression problem described in Section 2.2, where it was found that, when working in terms of an unnormalised variable, the effect on the values of the fitted model of rounding the computed parameter values was severe. If the data x-values for that data set are *normalised* as described above, and we use the same numerical algorithm for solving the polynomial regression problem, we obtain the following solution:

$$a_0 = 4.788111888112,$$
$$a_1 = 2.050000000000,$$
$$a_2 = 0.1660839160839,$$

where each value is quoted to thirteen significant figures. The parameters do not assume a wide range of values as was the case when working with an unnormalised variable (Section 2.2). Furthermore, the error in the model values over the range of the x-data incurred by rounding the parameter values to four digits after the decimal point (as was done in Section 2.2) is approximately 10^{-4}, and well within the measurement error for the original data.

In order to be able to work effectively with polynomial functions of arbitrary degree, it is recommended that *Chebyshev* polynomials (of the first kind) are used as basis functions for polynomials of a normalised variable [12]. The Chebyshev polynomials are defined by

$$T_j(X) = \cos\left(j \cos^{-1} X\right), \quad X \in [-1,+1],$$

and may be readily generated from the recurrence

$$T_0(X) = 1,$$
$$T_1(X) = X,$$
$$T_j(X) = 2XT_{j-1}(X) - T_{j-2}(X), \quad j = 2,3,\ldots$$

The Chebyshev polynomial $T_j(X)$ is a polynomial of degree j, and contains only odd powers of X if j is odd and only even powers if j is even. Moreover, it has the property of $\cos j\theta$ that it possesses a maximum and minimum value of +1 and −1, respectively. The functions $T_j(X)$ have much better properties than the monomials for modelling purposes. In particular, their shapes are "dissimilar", being near-orthogonal (the zeros of T_j and T_{j-1} interlace), and usually there is consequently little correlation between the fitted parameters. A polynomial $p(X)$ of degree n may be represented using the Chebyshev polynomials as the Chebyshev series

$$p(X) = {\sum_{j=0}^{n}}' a_j T_j(X), \quad X \in [-1,+1].$$

The prime on the summation sign indicates that the first term is to be halved, a convention that is common in work involving Chebyshev polynomials, and one that simplifies some of the calculations. The above Chebyshev series may be readily evaluated, differentiated and integrated using simple recurrence relations.

Example 2.

Consider the problem of a finding a best-fit sphere to measured data. A general parametrisation for the sphere is

$$A\left(x^2 + y^2 + z^2\right) + Bx + Cy + Dz + E = 0,$$

where (x, y, z) is a point on the sphere. Notice that if we multiply all the coefficients in this equation by a (non-zero) constant we do not change the sphere, and so to parametrise the sphere properly it is necessary to constrain the coefficients.

One possibility is to set $A = 1$. Then, the equation for the sphere may be re-arranged as

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2,$$

where

$$x_0 = -\frac{B}{2}, \quad y_0 = -\frac{C}{2}, \quad z_0 = -\frac{D}{2}, \quad r = \frac{\sqrt{B^2 + C^2 + D^2 - 4E}}{2}.$$

This corresponds to the (most straightforward) parametrisation of a sphere in terms of its centre coordinates (x_0, y_0, z_0) and radius r.

We expect the "centre/radius" parametrisation to be safe for representative data that covers a large portion of the sphere because for such data the "curvature" coefficient A will be non-zero and it is safe to apply the constraint $A = 1$. However, for data that is near-planar and only covers a small patch of the sphere, as may happen when measuring a lens, we expect A to be small compared to the other coefficients, and setting $A = 1$ is no longer a sensible way of parametrising a sphere for such data.

Now a plane such that its normal vector points approximately along the z-axis is described by the equation

$$\alpha x + \beta y + z = d.$$

It follows that for data that is near-planar measured on a sphere, with the plane such that its normal vector points along the z-axis, a suitable constraint for the sphere model is to set $D = 1$. If the data lies near to a different plane, we can always rotate the data so that the constraint $D = 1$ applies.

Finally, [13] discusses a parametrisation of spheres, based on the constraint

$$B^2 + C^2 + D^2 - 4AE = 1,$$

that can be applied in both circumstances of representative and near-planar data.

5. Sources of Scientific Software

See [14, 15]. The following lists are not exhaustive but indicative of sources of software that are relevant to metrology. They cover software packages, software libraries and distribution services providing generic mathematical functionality that may be useful in metrology. The items are given in no particular order and with no recommendation attached.

5.1 Software packages

Matlab, S-PLUS, MathCAD, Mathematica, Microsoft Excel, and LabVIEW are some of the software packages that provide mathematical and statistical functions and are used across metrology. Information concerning the assessment of the numerical performance of some of the functions provided by these software packages may be found in [16–24].

5.2 Software libraries

5.2.1 LAPACK

LAPACK [25] is a collection of Fortran subroutines for analysing and solving linear equations and linear least-squares problems, as well as solving eigenvalue and singular value matrix

problems. To a great extent, LAPACK supersedes the LINPACK [26] (Annex B) and EISPACK [27] (Annex C) libraries.

Linear equations and least-squares problems arise across metrology, especially in the context of *modelling* as follows. The replacement or representation of measured (discrete) data by a mathematical model is highly relevant to metrology. Typical objectives for such operations are data reduction and data smoothing, viz., noise-content reduction. The established model has many subsequent uses, including evaluation, differentiation and integration (e.g., for predicting derived quantities) as well as comparison (e.g., with a manufacturer's specification). Each measurement and the corresponding particularisation of a linear model for that measurement yields one equation in the linear system to be analysed. Linear systems also arise within the iterative solution of nonlinear modelling problems, where at each iteration the model is linearised about the current estimate of the solution (see Section 5.2.2).

The behaviour of measuring instruments can often be modelled in mathematical terms using differential or integral equations. For example, such models characterise the manner in which a physical quantity such as temperature or electric potential is distributed. Linear equations frequently arise from the application of finite element or boundary element methods to discretised versions of these equations.

Structure in the equations is commonplace and results from the nature of the measurement model, which may include *local* information (e.g., relating to points on an artefact to be calibrated) and *global* information (e.g., relating to self-calibration constants which apply to all the measurements). Software that exploits such structure permits large metrological models to be solved efficiently and robustly.

The eigen-analysis and singular value decomposition of matrices forms the backbone to much of the mathematical software used in metrology, including the analysis of linear and nonlinear systems of equations resulting from the interpolation, approximation or other modelling of measurement data, and to carry out perturbation and sensitivity studies of models of measurement systems. An eigen-analysis of the Jacobian matrix, i.e., the matrix of values of the model basis functions evaluated at the measurement points, is a powerful tool for investigating the properties of model parametrisations (e.g., the extent to which a model parametrisation gives a well-conditioned problem), as well as providing information to assist with experimental design (e.g., concerned with choosing measurement strategies).

The original goal of the LAPACK project was to make the widely used LINPACK (Annex B) and EISPACK (Annex C) libraries run efficiently on shared-memory vector and parallel processors. LAPACK routines are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subprograms (BLAS). While LINPACK and EISPACK are based on the vector operation kernels of the Level 1 BLAS [28], LAPACK was designed to exploit the Level 3 BLAS – a set of specifications for Fortran subprograms that do various types of matrix multiplication and the solution of triangular systems with multiple right-hand sides. Because of the coarse granularity of the Level 3 BLAS operations, their use promotes high efficiency on many high-performance computers, particularly if specially coded implementations are provided by the manufacturer. However, a model Fortran implementation of the BLAS is available from *netlib* (see Section 5.3.2) in the BLAS library, and this allows LAPACK to be used these days on most modern computers.

5.2.2 MINPACK

MINPACK [29] is a collection of Fortran subprograms for the solution of systems of nonlinear equations and nonlinear least squares problems. Such nonlinear equations and least-squares problems arise across metrology, especially in the context of discrete modelling (as described in Section 5.2.1) in cases where the metrology system is described by a *nonlinear* model. Each equation in the system of equations corresponds to a measurement and the corresponding particularisation of the *nonlinear* model. In addition, in measurement applications where more

than one variable is subject to measurement error (e.g., both control and response variables are measured inexactly), the corresponding model equations give rise to nonlinear systems of equations to be solved.

The MINPACK package includes facilities for approximating Jacobians using finite differences, for systems of equations with banded Jacobians, for least squares problems with large amounts of data, and for checking the consistency of a user-supplied Jacobian with the functions supplied.

5.2.3 NAG

The NAG Fortran library [30] is a comprehensive collection of over 1000 routines for performing computations in the areas of

- elementary and special functions,
- linear algebra,
- interpolation,
- solution of nonlinear equations,
- optimisation,
- differentiation and integration,
- differential and integral equations,
- integral transforms,
- approximation,
- statistics and probability,
- data handling,
- computational geometry, and
- graphics,

many of which have application across metrology. The library provides facilities for solving algebraic and differential equations, optimisation and approximation, time series analysis, the statistical analysis of data, and many other areas. In the area of statistics and probability, for example, tools are included for the calculation of basic statistical distributions (e.g., the *t*, *F* and χ^2 statistics, used in tests associated with normally distributed data, and the determination of confidence intervals), random number generation (used, e.g., in simulating measurement systems), and non-parametric statistics (e.g., that avoid the need to make traditional assumptions about a measurement system such as normality and linearisability).

A subset of the library is available in the form of *dynamic linked libraries* (DLLs) that enable the software to be integrated seamlessly with a variety of applications, including Visual Basic, Visual Basic Applications, Excel, Microsoft C++, Borland C++, Borland Delphi, PowerBuilder and Watcom C/C++. Many of these applications are used within metrology for data logging and instrument control. The provision of the library in this form gives the metrologists direct access to reliable and robust software within the measurement environment.

5.2.4 IMSL

The IMSL Fortran library [31] is a collection of Fortran subroutines for solving a variety of problems in the areas of linear systems, eigensystem analysis, interpolation and approximation, integration and differentiation, differential equations, nonlinear equations, optimisation, special functions and statistics and probability.

5.2.5 DASL

The National Physical Laboratory's Data Approximation Library (DASL) [32] is a coordinated library of Fortran subroutines for fitting data representing relationships between two or more variables.

DASL caters particularly for situations in which the mathematical form of the relationship is unknown. Two types of fitting functions are available: polynomials and splines (or piecewise polynomials) that may be used to define mathematical curves representing relationships between variables that are explicit, periodic and parametric. DASL also enables users to provide their own types of function for fitting data, provided the unknown coefficients occur linearly in the function.

The main operations undertaken are *least-squares approximation* and *interpolation*. Having employed the library to obtain a satisfactory curve or surface, DASL can then be used to perform various subsequent calculations including *evaluation, differentiation, integration* and the evaluation of *statistics*.

5.2.6 NOSL

The National Physical Laboratory's Numerical Optimisation Subroutine Library (NOSL) [33] is a library of Fortran subroutines for solving a variety of optimisation problems. Many metrological processes can be expressed in the form of an algebraic model relating (a) the measured data, (b) the metrological quantities (or parameters) to be determined, and (c) the measurement errors. The model may be complicated, with many parameters, and contain "side conditions" relating to physical properties or feasibility. Solution of the model, i.e., determination of the metrological parameters, can be approached through the use of numerical optimisation software. Typical objectives of optimisation in metrology are "best-fit" criteria for solving models of measurement systems, the optimal location of measurement points (experimental design), and the design of instruments to have some optimal property.

NOSL covers problems including unconstrained minimisation of general functions of one or more variables, unconstrained non-linear least-squares problems, and minimisation subject to constraints that may take the form of simple bounds, linear and non-linear constraints.

5.2.7 NPLFIT

The National Physical Laboratory has developed a software package for fitting calibration curves to experimental data. The package provides a coherent and usable interface to curve-fitting software representing current best practice, and may be used for exploratory data analysis or production purposes, e.g., for quantifying difficult data behaviour or for producing calibration curves and information derived from them for calibration certificates.

The use of the package enables two important classes of *empirical* models, viz., polynomials and polynomial splines, to be fitted to data. (The package is not designed for fitting *physical* models to data.) The fitting criteria available are

- 2-norm (least-squares) for data with Gaussian and other symmetric error distributions,
- 1-norm (min-sum-mod) for data with long-tailed error distributions and wild points, and
- infinity-norm (min-max) for data with rectangular error distributions (such as correctly rounded tabular data).

Easily used facilities are provided for selecting that model from a family of (polynomial or spline) models that is most appropriate for the data. The package also enables the fitted function to be evaluated, differentiated and integrated. Further, for least-squares fits, the standard uncertainties of values, derivatives, integral, and other derived quantities, taking full account of correlations, can be estimated in accordance with the ISO Guide to the Expression of Uncertainty in Measurement [34] and UKAS Document M 3003 [35].

Many of the algorithms underpinning the software are provably the most numerically stable available, and thus permit the package to be used for data that is arbitrarily scaled, shifted or normalised. Moreover, fits by polynomials of high degree or splines with large numbers of knots can reliably be obtained. In particular, the "best fit" in terms of polynomial degree or number of spline knots can be identified as indicated above.

Although currently restricted to use within the National Physical Laboratory, it is intended to make the NPLFIT software package more widely available through the METROS distribution service (see Section 5.3.1).

5.3 Distribution services

5.3.1 METROS

The METROS system [36, 37] designed jointly by the National Physical Laboratory and the Numerical Algorithms Group Ltd, and being developed by the National Physical Laboratory (as "EUROMETROS") under a EUROMET project, allows metrologists to gain access to scientific software necessary for their work. The central idea is that of the *key function*. This is either a function that may be used in many different applications or a function that is seen as being fundamental to an area of metrology. A list of key functions is given in [36], and this represents an attempt to list the fundamental computational requirements of metrologists. Most metrologists may not be familiar with these functions but will view their problem from an applications perspective. The METROS system therefore contains *application functions* based on key functions and links between these application functions and the key functions. The software provided by METROS is available in a variety of different computing environments and platforms, and is supported by appropriate *case studies* and *best practice guides* which will also be contained in the METROS system.

For more information on the METROS system, see

http://www.npl.co.uk/ssfm/metros/index.html

5.3.2 GAMS

The Guide to Available Mathematical Software (GAMS) project of the National Institute of Standards and Technology (NIST) studies techniques to provide scientists and engineers with improved access to reusable computer software which is available to them for use in mathematical modelling and statistical analysis. One of the products of this work is an on-line cross-index of available mathematical software. This system also operates as a virtual software repository. That is, it provides centralised access to such items as abstracts, documentation, and source code of software modules that it catalogues; however, rather than operate a physical repository of its own, this system provides transparent access to multiple repositories operated by others.

Currently four software repositories are indexed: three maintained for use by NIST staff (but accessible to the public), and *netlib*, a publicly accessible software collection maintained by Oak Ridge National Laboratory and the University of Tennessee at Knoxville and Bell Labs (see http://www.netlib.org/). This represents some 10,000 problem-solving modules from more than 100 software packages. The vast majority of this software represents Fortran subprograms for mathematical problems which commonly occur in computational science and engineering, such as solution of systems of linear algebraic equations, computing matrix eigenvalues, solving nonlinear systems of differential equations, finding minima of nonlinear functions of several variables, evaluating the special functions of applied mathematics, and performing nonlinear regression. Among the packages catalogued are: the IMSL, NAG, PORT, and SLATEC libraries; the BLAS, EISPACK, FISHPAK, FNLIB, FFTPACK, LAPACK, LINPACK, and STARPAC packages; the DATAPLOT and SAS statistical analysis systems; as well as other collections such as the Collected Algorithms of the ACM (Association of Computing Machinery). Note that although both public domain and proprietary software is

catalogued here, source code of proprietary software products are not available, although related items such as documentation and example programs often are.

All catalogued problem-solving software modules are assigned one or more problem classifications from a tree-structured taxonomy of mathematical and statistical problems. Users can browse through modules in any given problem class. To find an appropriate class, one can utilise the taxonomy as a decision tree, or enter keywords that are then mapped to problem classes. Search filters can be declared which allow users to specify preferences such as computing precision or programming language. In addition, users can browse through all modules in a given package, all modules with a given name, or all modules with user-supplied keywords in their abstracts.

At its highest level, the GAMS classification system contains the following classes:

A Arithmetic and error analysis

B Number theory

C Elementary and special functions

D Linear algebra

E Interpolation

F Solution of nonlinear equations

G Optimisation

H Differentiation and integration

I Differential and integral equations

J Integral equations

K Approximation

L Statistics and probability

M Simulation and stochastic modelling

N Data handling

O Symbolic computation

P Computational geometry

Q Graphics

R Service routines

S Software development tools

Z Other

For more information on the GAMS project, see

http://gams.nist.gov

6. Conclusion

In this report we have provided information and guidelines to help users select and use software in such a way that it is fit for purpose for their metrology applications, where fitness for purpose is defined in terms of meeting requirements relating to the numerical accuracy of the results returned by the software.

We have discussed measures of fitness for purpose in this regard, and discussed some of the factors that affect how users might decide fitness for purpose requirements accounting for the possible ways the results returned by scientific software may be subsequently used. We have also presented approaches to verifying or testing the numerical accuracy of scientific software, and provided guidance on how users might make better use of existing software through pre-processing the input data to such software and careful choice of model parametrisation. Finally, details of software and sources of software providing functionality that is relevant to metrology applications have been listed.

7. Acknowledgement

This report constitutes one of the deliverables 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.

Appendix A References

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

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

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

[4] H.R. Cook, M.G. Cox, M.P. Dainton and P.M. Harris. *A methodology for testing spreadsheets and other packages used in metrology*. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CISE 25/99, September 1999.

[5] 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.

[6] 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.

[7] J. Rogers, J. Filliben, L. Gill, W. Guthrie, E. Lagergren and M. Vangel. StRD: Statistical reference datasets for assessing the numerical accuracy of statistical software. *NIST TN1396*, National Institute of Standards and Technology, US.

[8] M.G. Cox and A.B. Forbes. Strategies for testing form assessment software. *NPL Report DITC 211/92*, National Physical Laboratory, 1992.

[9] A.B. Forbes and P.M. Harris. A comparison of methods used for the calculation of effective area in the calibration of pressure balances. *NPL Report CISE 2/95*, National Physical Laboratory, 1995.

[10] 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.

[11] 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.

[12] L. Fox and I.B. Parker. *Chebyshev Polynomials in Numerical Analysis*. Oxford University Press, 1968.

[13] A.B. Forbes. Robust circle and sphere fitting by least squares. *NPL Report DITC 153/89*, National Physical Laboratory, 1989.

[14] A B Forbes. Mathematical software for metrology: meeting the needs of the metrologist. *Advanced Mathematical Tools in Metrology* (P Ciarlini, M G Cox, R Monaco and F Pavese, Eds.), World Scientific, Singapore, pp 247−254, 1994.

[15] J. du Croz. Relevant general-purpose mathematical and statistical software. *Advanced Mathematical Tools in Metrology II* (P Ciarlini, M G Cox, F Pavese and D Richter, Eds.), World Scientific, Singapore, pp 22–28, 1996.

[16] 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.

[17] J. Barrett and M.P. Dainton. *Testing spreadsheets and other packages used in metrology: Testing the intrinsic functions of MathCAD*. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CMSC 05/00, September 2000.

[18] J. Barrett and M.P. Dainton. *Testing spreadsheets and other packages used in metrology: Testing the intrinsic functions of S-PLUS*. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CMSC 06/00, September 2000.

[19] M.G. Cox, M.P. Dainton and P.M. Harris. *Testing spreadsheets and other packages used in metrology: Testing functions for the calculation of standard deviation*. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CMSC 07/00, October 2000.

[20] M.G. Cox, M.P. Dainton and P.M. Harris. *Testing spreadsheets and other packages used in metrology: Testing functions for linear regression*. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CMSC 08/00, October 2000.

[21] L. Knusel. On the accuracy of statistical distributions in Microsoft Excel 97. *Computational Statistics and Data Analysis*, **26**(3), pp 375-7, 1998.

[22] B.D. McCullough. Assessing the reliability of scientific software: Part I. The Amer. Statist., **52**(4), pp 358–366, 1998.

[23] B.D. McCullough. Assessing the reliability of scientific software: Part II. The Amer. Statist., **53**(2), pp 149–159, 1999.

[24] B.D. McCullough and B Wilson. *On the accuracy of statistical procedures in Microsoft Excel 97*. Computational Statistics and Data Analysis **31**, pp 27–37, 1999.

[25] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen. *LAPACK Users Guide*. SIAM, Philadelphia, Second Edition, 1994.

[26] J.J. Dongarra, J.R. Bunch, C.B. Moler and G.W. Stewart. *LINPACK Users' Guide*. SIAM, Philadelphia, 1979.

[27] B.T. Smith, J.M. Boyle, B.S. Garbow, Y. Ikebe, V.C. Klema and C.B. Moler. *Matrix Eigensystem Routines – EISPACK Guide*. Springer-Verlag, New York, 1974.

[28] C. Lawson, R. Hanson, D. Kincaid and F. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software*, **5**, pp 308–323, 1979.

[29] B.S. Garbow, K.E. Hillstrom and J.J. Moré. *User's Guide for MINPACK-1*. Technical Report ANL–80–74, Argonne National Laboratory, 1980.

[30] NAG. *The NAG Fortran Library*. The Numerical Algorithms Group Ltd., Wilkinson House, Jordan Hill Road, Oxford, UK.

[31] IMSL. *IMSL User's Manual*. IMSL Inc., Houston, Texas, 1987.

[32] G.T Anthony and M.G. Cox. The National Physical Laboratory's Data Approximation Subroutine Library. In J.C. Mason and M.G. Cox, editors, *Algorithms for Approximation*, pp 669–687, Clarendon Press, 1987.

[33] S.M. Hodson, H.M. Jones and E.M.R. Long. A brief guide to the NPL Numerical Optimization Software Library. *National Physical Laboratory*, 1993.

[34] ISO. *Guide to the Expression of Uncertainty in Measurement*. International Organization for Standardization, ISBN 92-67-10188-9, 1993.

[35] UKAS. *M 3003: The Expression of Uncertainty and Confidence in Measurement*. 1997.

[36] R. Barker and G. Morgan. *Design of the metrology software environment (METROS)*. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CISE 43/00, February 2000.

[37] R. Barker and G. Morgan. *Specification of the METROS key functions*. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CISE 44/00, February 2000.

Appendix B LINPACK

LINPACK [27] is a collection of Fortran subroutines for analysing and solving linear equations and linear least-squares problems. The LINPACK package may be used to solve linear systems whose matrices are general, banded, symmetric indefinite, symmetric positive definite, triangular, and tridiagonal-square. In addition, the package computes the QR and singular value decompositions of rectangular matrices and applies them to least-squares problems. LINPACK uses column-oriented algorithms to increase efficiency by preserving locality of reference. LINPACK was designed for supercomputers in use in the 1970s and early 1980s, and has been largely superseded by LAPACK (Section 5.2.1).

Appendix C EISPACK

EISPACK [28] is a collection of Fortran subroutines for computing the eigenvalues and eigenvectors of matrices. Nine classes of matrices are covered by the EISPACK package: complex general, complex Hermitian, real general, real symmetric, real symmetric banded, real symmetric tridiagonal, special real tridiagonal, generalized real, and generalized real symmetric matrices. In addition, two routines are included that use singular value decomposition to solve certain least-squares problems. EISPACK has been superseded for the most part by LAPACK (Section 5.2.1).