# The design and use of reference data sets for testing scientific software[1]

M. G. Cox* and P. M. Harris

Centre for Information Systems Engineering, National Physical Laboratory, Teddington, Middlesex TW11 0LW, UK

*Fax: 0181 977 7091; E-mail: Maurice.Cox@npl.co.uk

June 26, 1998

**Abstract**

A general methodology for evaluating the accuracy of the results produced by scientific software has been developed at the National Physical Laboratory. 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. 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.

In this paper it is shown how reference data sets can be designed for testing software implementations of solutions to a broad class of problems arising throughout science. It is shown how these data sets can be used in practice and how the results provided by software under test can properly be compared with reference results. The approach is illustrated with three examples: (i) mean and standard deviation, (ii) straight-line fitting, and (iii) principal-components analysis. Software for such problems is used routinely in many fields, including optical spectrometry.

## 1. Introduction

There is a growing need to ensure 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. To this end the National Physical Laboratory has developed a general methodology which is applicable to a range 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 allows for 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 comparison of the results returned by the test software with the reference results is made objective by the use of quality metrics that account for the absolute differences between the test and reference results, the computational precision of the underlying hardware, and the "degree of difficulty'' of the data set.

---

[1] Presented at the *International Conference of Optical Spectrometry: Applications and Instrumentation into the 21st Century*, Royal Holloway College, UK, 28 June - 2 July 1998. To appear in the journal *Analytica Chimica Acta*.

The methodology has been applied in the National Physical Laboratory's own software development, and to test specific parts of proprietary software packages, including spreadsheets, in common use.

In this paper it is shown how reference data sets can be designed for testing software implementations of solutions to a broad class of problems arising throughout science. It is shown how these data sets can be used in practice and how the results provided by software under test can properly be compared with reference results. The approach is illustrated with three examples: (i) mean and standard deviation, (ii) straight-line regression, and (iii) principal-components analysis. Software for such problems is used routinely in many fields, including optical spectrometry.

## 1.1 Why software fails or produces unreliable results

Software used in scientific disciplines can be unreliable because it implements numerical algorithms that are unstable or not robust. Some of the reasons [1] 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],
2. the use of an unstable parametrisation[3] of the problem,
3. the use of a solution process that exacerbates the inherent (natural) ill-conditioning of the problem[4], and
4. a poor choice of formula from a set of mathematically (but not numerically) equivalent forms.

## 1.2 Requirements for testing: unambiguous specification

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 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.

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.

---

[2] A simple instance of data transformation is "coding", as used traditionally in many statistical studies. A data value $x$ is replaced by a value $x' = Ax + B$ which has been scaled (by $A$) and translated (by $B$) such that all such $x$-values lie in a normalised range such as -1 to +1.

[3] Problem parametrisation is considered in Section 2.3

[4] The *condition* of a problem is a measure of the sensitivity of the solution to changes in the "input data". A well-conditioned problem has the property that a small change to the input data results in a correspondingly small change in the solution (when solved using an optimally-stable algorithm; see Section 3.1), whereas for an ill-conditioned problem the results would change appreciably. A poor algorithm can introduce the effects of ill-conditioning, even if not present in the problem. Moreover, it can worsen any ill-conditioning effects already present.

### 1.3 Black-box testing and reference data sets

Although scientific software is very widely used, it is rarely tested in an objective and impartial manner. Some approaches for such testing have been developed, however. These address the individual software modules called directly by scientists or as part of software packages used in science [2, 3, 4]. 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.

The concern here is with reference data sets for the black-box testing of scientific software. Such testing involves submitting reference data sets to software under test, the *test software*, and the resulting output, the *test results*, is compared with known reference results. For the application of this approach, it is necessary to be able to generate *reference pairs*, i.e., reference data sets *and* the corresponding reference results.

Section 2 of this paper addresses some of the main considerations in designing reference data sets and Section 3 the use of these data sets. Section 4 is concerned with the production of reference data sets in the important case of least-squares regression. Section 5 presents examples of application. These are (i) mean and standard deviation (the problem of determining which can be posed as fitting a constant function to data), (ii) straight-line regression (the "next-hardest'' problem following fitting a constant), and (iii) principal-components analysis. Section 6 contains concluding remarks.

Some technical terms are used in this paper which may not be familiar to the intended audience. Informal definitions of these terms are included as footnotes.

## 2. Design of reference data sets

Some of the main considerations in designing reference data sets for checking the fitness for purpose of software are

1. the identification of possible deficiencies in the test software, such as severe rounding-error effects, subtractive cancellation[5] and overflow[6] in the computation,
2. the incorporation of known properties in the data sets, such as the "degree of difficulty''[7] of the associated problem, and
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.

In principle, the approach outlined here is capable of application to a very general class of problems, viz., the determination of a minimum of a general function of any number of variables. Here, however, the more modest class of problems of least-squares regression[8] is considered. This class contains the problem of computing the arithmetic mean and standard deviation of a data set.

[5] An example of subtractive cancellation is given in Section 5.1

[6] Overflow is the consequence of performing an arithmetic operation such that the result is too large (in magnitude) to be held within the arithmetic system employed. Similarly, underflow can arise; the result is too small to be represented and is replaced by zero or an approximation.

[7] The degree of difficulty of a problem is closely related to *condition* (Section 1.1) and also incorporates the effect of problem scaling (Section 3.1).

[8] Least-squares regression is the fitting of a model to data using the method of least squares.

## 2.1 Fitness for purpose

Software can at least in part be checked that it is fit for purpose by ensuring that it performs properly for reference data sets that are *representative* of the application. It can further be so checked by including data sets designed to reveal the extent to which the software can yield sufficiently accurate solutions. For both these purposes it is highly desirable that the data sets possess certain properties. One valuable property is that the problems identified by the data sets possess *known* solutions.

Regression software can be deficient in diverse ways, such as an inability to deal accurately with observational data containing a significant noise component, large numbers of observations, or certain distributions of points. Suitable tests can be devised to help reveal such deficiencies.

## 2.2 Design based on prior knowledge of solution

The approach adopted at the National Physical Laboratory to the design of reference data sets is based on the *characterisation* of the solution to the problem addressed by the software. For example, a necessary condition [5] for a solution of a nonlinear least-squares problem is that $J^{\mathrm{T}}\mathbf{e} = 0$. Here $\mathbf{e}$ is the vector of residuals and $J$ is the Jacobian matrix, both evaluated at the solution. ($J$ is the rectangular matrix of partial derivatives of the algebraic expressions for the residuals with respect to the problem parameters, evaluated for each data point.) Note that $J^{\mathrm{T}}J$ is the *normal matrix*, i.e., the matrix occurring in the normal-equations approach [6, p237] to regression.

Reference data sets are constructed, using a *data generator*, to have *known* solutions, i.e., solutions specified *a priori*. The use of data generators is illustrated in Section 4.1.

There is an alternative approach to employing a data generator, viz., to use *reference software*. Such software is 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 [3], 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).

There is a further significant advantage of the use of data generators over that of reference software. The latter can of course be used to furnish the solution corresponding to any data set within its domain of applicability[9]. However, in the case of the former, required specific properties can be built in, e.g., in polynomial regression a data set for which a particular high-degree polynomial would be required.

Finally, real measurement data sets often have certain properties that in total can be quite difficult to characterise. However, data sets can be devised which are "close'' to such data sets *and* which correspond to known solutions [7].

## 2.3 Problem parametrisation

Problem parametrisation can readily be demonstrated using 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

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

[9] The domain of applicability of an item of software is the set of inputs that the software has been designed to accept and for which it should produce reliable results.

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 [2, 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 Standards Organisation [8] 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. However, within the area of regression, the residuals of a correctly computed solution are invariant to the parametrisation and are often suitable for testing purposes. Section 4.1 considers the use of residuals in testing straight-line regression software.

# 3. Use of reference data sets

This section treats the factors influencing the comparison of reference results and test results, and indicates the quality metrics used for this purpose. For further details, see [2].

Major factors affecting the comparison of test and reference results are the computational precision $\eta$ of the arithmetic used to produce the test results[10], the numerical precision of the reference data sets, the degree of difficulty $K$ (including the scale) of the problem, and the problem parameterisation. The effects of $\eta$ and $K$ are considered here, assuming that problem-scale effects have been incorporated into $K$ and that reference results are known to adequate accuracy for comparison purposes. Further information concerning these factors and their influence and other effects such as input-output is available [2, 9].

## 3.1 Quality metrics

After the test software has been applied to each reference data set, (at least) three quality metrics which indicate how the test software has performed on that data set can be determined [2]:

1. The difference between the test and reference results, an *absolute* measure of departure,

2. The number of figures of agreement, a *relative* measure of departure, and

3. A *performance measure* depending on the major factors above.

The use of the third metric is considered here. It is indicated below how it is determined and used in practice.

Express the reference results and test results as vectors of floating-point numbers. Let $\mathbf{d} = \mathbf{b}^{(\text{test})} - \mathbf{b}^{(\text{ref})}$, where $\mathbf{b}^{(\text{test})}$ denotes the test results and $\mathbf{b}^{(\text{ref})}$ the reference results. $d(\mathbf{b}) = \text{RMS}(\mathbf{d})$ is used here to measure this difference, where $\text{RMS}(\mathbf{x}) = \|\mathbf{x}\|_2/\sqrt{n}$ denotes the root-mean-square value of an $n$-vector $\mathbf{x}$.

For each data set the problem *degree of difficulty K* is determined such that $d(\mathbf{b})$ can sensibly be compared with a tolerance equal to $K\eta$. (The determination of $K$ would normally be carried out by a numerical analyst.) Thus, $K$ depends on the problem condition and scale[11]. The quantity $d(\mathbf{b})/(K\eta)$ will then have the property that it is of order unity for software that performs near-optimally and is possibly very large for poorly-performing software.

The performance measure $P(\mathbf{b})$ is defined by

$$P(\boldsymbol{b}) = \log_{10}\left(1+\frac{d(\boldsymbol{b})}{K\eta}\right). \tag{3}$$

It indicates the number of figures of accuracy *lost* by the test software over and above what software based on an optimally stable algorithm[12] would produce, being near zero if $\mathbf{b}^{(\text{test})}$ and $\mathbf{b}^{(\text{ref})}$ agree as closely as could be expected, and having a value of about $k$ if the agreement is $k$ figures less than this. A related performance measure is used in testing software for evaluating special functions [4].

# 4. Production of reference data sets

This section is concerned with the use of data generators to produce reference data sets in regression and related areas, leading to the *null-space* approach to data generation (based on solution characterisation). It also covers graded data sets and performance profiles. The approach is applicable to all aspects of least-squares model fitting with physical or empirical models, data with constant or non-constant variance, and data with or without correlated errors. Here, we only consider the simplest case.

## 4.1 Reference software and data generators

A *reference pair*, i.e., a reference data set and the corresponding reference results, may be produced in two ways (Section 2.2):

1. Start with a reference data set and apply reference software (as, e.g., in [10]) 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.

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 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. We return to this example in Sections 4.2 and 5.1.

Null-space methods [9, 7] provide a facility for generating *families* or *classes* of data sets. 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.

---

[10] For the very commonly used floating-point arithmetic, $\eta$ 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.

[11] The scale of a problem is taken into account through the dimensions, physical or otherwise, of the results. It is essential that the expression of the same results in different units, such as kilograms or micrograms, does not adversely influence the perception of the behaviour of the software.

[12] 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.

The null-space approach is now illustrated for the linear least-squares model $\mathbf{y} = J\mathbf{a} + \mathbf{e}$, where $J$ denotes the observation (Jacobian) matrix, $\mathbf{y}$ the observation vector, $\mathbf{a}$ the vector of model parameters and $\mathbf{e}$ the vector of residuals. For example, if the data to be fitted is $\{(x_i, y_i)\}$, $i = 1,\ldots, m$, and the model is the straight line with parametrisation (1), then

$$J = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \quad \mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_m \end{bmatrix}, \quad \boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}.$$

Now the least squares solution is characterised by $J^{\mathrm{T}}\mathbf{e} = 0$ [6]. This equation implies two conditions, one that

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

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

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

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 $J^{\mathrm{T}}$, i.e., $J^{\mathrm{T}}N = 0$.[13] 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 $\mathbf{a}$ *unchanged*. This result can be seen as follows. The condition $J^{\mathrm{T}}\mathbf{e} = \mathbf{0}$ and the model $\mathbf{y} = J\mathbf{a} + \mathbf{e}$ imply the *normal equations* $J^{\mathrm{T}}J\mathbf{a} = J^{\mathrm{T}}\mathbf{y}$. But, from the definition of the null space, $J^{\mathrm{T}}\mathbf{r} = J^{\mathrm{T}}N\mathbf{u} = (J^{\mathrm{T}}N)\mathbf{u} = \mathbf{0}$. Thus, from any one data set any number of further data sets having the same solution can readily be constructed by choosing vectors $\mathbf{u}$.

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 $J$ can be calculated very reliably using the singular-value decomposition [6, p602]. One such implementation is the `null` function in Matlab [11]. Additionally, it is possible to check the accuracy of the generated data sets by forming $J^{\mathrm{T}}\mathbf{e}$ and assessing the closeness of $\|J^{\mathrm{T}}\mathbf{e}\|_2$ to zero, relative to the value of $\|J^{\mathrm{T}}\|_2 \, \|\mathbf{e}\|_2$. This check is generic and does not require the availability of software to solve the particular problem considered.

## 4.2 Arithmetic mean and standard deviation

The arithmetic mean and standard deviation of a set of numbers are considered as an illustrative example in Section 5.1, but here we indicate how the above concepts may be applied in this simple case. This example is closely related to the above straight-line model, since determining the mean of a set of numbers is equivalent to fitting them by a constant, and their standard deviation is identical to the root-mean-square residual of the fit. Any two values having $\bar{x}$ as their mean are given by $\bar{x} \pm \lambda$ for any $\lambda$. (This is the only null-space perturbation possible in this case.) Further, any two values given by $\mu \pm s/\sqrt{2}$ have $s$ as their (sample) standard deviation for any value $\mu$. Thus, by varying $\mu$, for example, different data sets of size two can be generated, where in each case the solution standard deviation is $s$.

[13] This basis can be expressed as an independent set of 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.

These data sets contain only two values, and in each case there are just *two* solution parameters, $\bar{x}$ and *s*. Null-space methods [7] permit the above concept to be extended to any number of values. Moreover, as intimated earlier, they permit not just simple problems such as that above to be handled, but also extend to very complicated problems, even involving nonlinear optimization with constraints[14].

In order to emphasise the importance of testing standard-deviation software, we comment that a particular spreadsheet package loses 12 of the 14 decimal figures available when $s = 0.1$ and $\mu = 10^6$, whereas another loses just three figures. When $\mu$ is increased to $10^7$, the first spreadsheet package returns *zero*(!) as the answer (as a result of losing *all* figures), whereas the second delivers a value differing from the true value by less than $10^{-9}$. Thus, $\mu$ (or $\lambda$) can be used as a *performance parameter*, i.e., a varying parameter to investigate performance.

### 4.3 Graded data sets and performance profiles

Software used in scientific disciplines often has the property that it works well for certain data sets but produces unacceptable results for other sets. There is therefore a danger that limited testing might not reveal its deficiencies. An example is polynomial regression software, where the results of fitting a polynomial of low degree can be perfectly acceptable, whereas those for higher degrees would be contaminated by errors introduced by the solution algorithm. In Section 5.2 the simplest polynomial (apart from the constant), the straight line, is again considered.

The testing of software on a small number of data sets can provide valuable information about the software, particularly if the sets are selected to be as disparate as possible in terms of their locations in the space of possible data inputs. Such data sets provide "spot checks" of the test software, and are useful in the sense that the software can be regarded as deficient if it fails to perform adequately on any one of them. They are also useful for the software developer to test, e.g., particular paths through the software. They are less effective for black-box testing where without additional information little can be assumed about the algorithms that have been implemented. Data sets of this form have been used for a number of years in testing software used in industrial inspection for fitting mathematical features such as spheres, cylinders and cones to data acquired using co-ordinate measuring machines [12] and will become more formally established when an ISO standard is published [8], and in chemometrics [2, 13].

*Graded* data sets can be used to help quantify the behaviour of scientific software. A *sequence* of data sets is prepared, where each data set in the sequence represents a problem that in some sense is more difficult than the previous member in the sequence. Such a suite of graded data sets is used to explore *one dimension* of the problem addressed by the software. The resulting *performance profile*, the *graph* of the values of *P* against the degree of difficulty *K* (or some related performance parameter), can provide valuable information about the software. If, for *graded reference data sets* (taken in order), the values of the resulting performance measure *P*(**b)** have a tendency to *increase*, it is likely that an unstable method has been employed. (Performance profiles in a somewhat different setting have also been studied by Lyness [14].) The 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 the following section.

## 5. Examples of application

We give three examples of application, the first two of which arise in virtually all fields of science, and the third in areas such as optical spectrometry and analytical chemistry. Although these examples are elementary, they are sufficient to illustrate the key aspects of software testing using the presented methodology.

[14] Constrained non-linear optimization is the problem of minimizing a mathematical function of a number of variables subject to a set of mathematical equalities and inequalities involving these variables [5].

## 5.1 The sample arithmetic mean and standard deviation

The *arithmetic mean* of $m$ numbers $x_i$, $i = 1, \ldots, m$, is defined by

$$\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i . \tag{4}$$

Apart from exceptional cases, most software for computing $\bar{x}$ from given $x_i$ will produce a result that has a relative accuracy approaching the computational precision $\eta$ and hence be acceptable in all application areas. The quality of software for computing the standard deviation is in comparison very varied across implementations. In fact, for some implementations, the software degrades rapidly in its performance for a sequence of graded data sets. The sample standard deviation $s$ is given (stably) by

$$s = \left\{ \frac{1}{m-1}\sum_{i=1}^{m} (x_i - \bar{x})^2 \right\}^{1/2} , \tag{5}$$

where $\bar{x}$ is the arithmetic mean defined in (4). Some software uses the *mathematically equivalent* (but unstable) form

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

It 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], $\bar{x}$ = 1.00, $\sum_i x_i^2$ = 5.0010, $m\bar{x}^2$ = 5.0000, and hence three to four significant figures are lost due to subtractive cancellation in forming $\sum_i x_i^2 - m\bar{x}^2$ = 5.0010 - 5.0000 = 0.0010. In some cases, e.g., a series of accurate replicate weighings of a standard kilogram to within 1mg, $\sum_i x_i^2$ will be *equal* to $m\bar{x}^2$ to all figures held on the computer. It is even possible that the computed value of $\sum_i x_i^2$ 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 $\sum_i x_i^2 - m\bar{x}^2$ 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 [13] of various software packages and electronic calculators, not one of the eight tested on weighing data produced the correct value of $s$ to two correct figures, although a perfectly acceptable value would have been obtained by the use of formula (5).

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 $\mu$ 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 $N = 60$. A value of the degree of difficulty $K$ was calculated in each case as the inverse coefficient of variation $\bar{x} / s$ [2]. Software implementations of the above two formulae were applied to these data sets and a performance profile determined. See Figure 1.

It is observed in Figure 1 that the stable algorithm (points joined by straight-line segments to give an essentially horizontal line) performs completely satisfactorily for problem degrees of difficulty *K* spanning 10 decades. 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 *K*. (This result is predicted by a detailed floating-point error analysis of formula (6).) A performance profile for standard-deviation software similar in behaviour to the generally rising line in Figure 1 could be taken as an indication that the unstable formula (6) (or a comparable formula) had been used. It is to be noted that, for almost 20% of the 60 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.

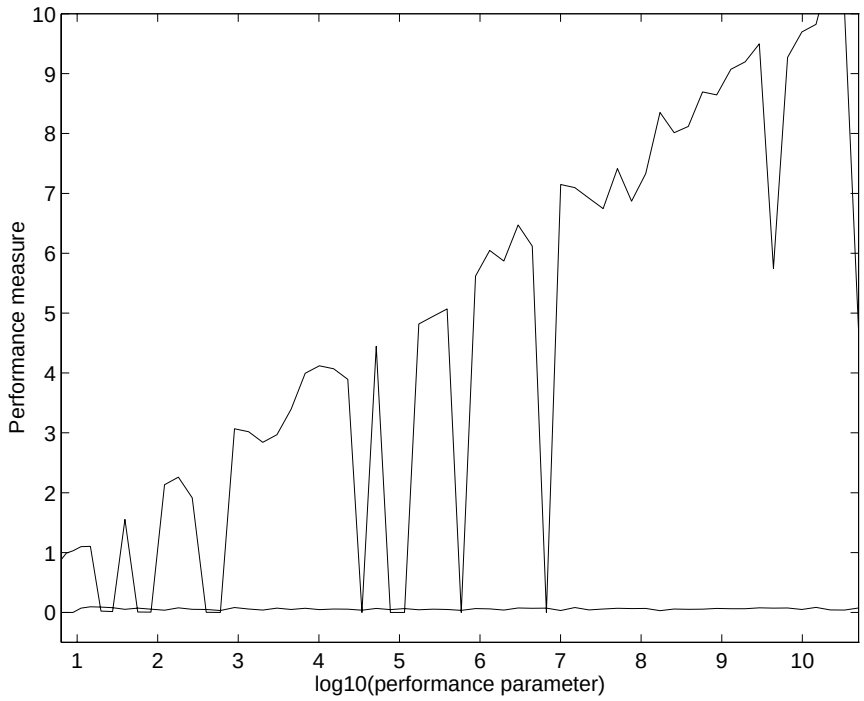


Figure 1: Performance profile for sample standard deviation against inverse coefficient of variation calculated using a stable algorithm (generally-horizontal line) and an unstable algorithm (generally-increasing line).

## 5.2 Straight-line regression

Polynomial-regression software can be deficient in a number of respects. Prime causes [2] are

1. The parametrisation of the polynomial (i.e., the choice of basis functions (monomials, Chebyshev polynomials[15], etc.) for representing the polynomial),
2. The scaling or normalisation of the problem data, and
3. The algorithm used to solve the linear algebraic equations defining the polynomial coefficients.

[15] The monomial representation of a polynomial $p(x)$ is $a_0 + a_1 x + a_2 x^2 + \ldots$. Chebyshev polynomials provide another way of expressing $p(x)$, viz., $a_0 T_0(x) + a_1 T_1(x) + a_2 T_2(x) + \ldots$, for different coefficients. Here, $T_j(x)$ is a polynomial of degree $j$ in $x$, given by $T_0(x) = 1$, $T_1(x) = x$, $T_j(x) = 2x T_{j-1}(x) - T_{j-2}(x)$, $j > 1$ [15].

Failure to scale or normalise the data sensibly or a poor choice of parametrisation can cause the resulting equations defining the parameters of the polynomial to be unnecessarily ill-conditioned. A poor choice of algorithm to solve this system can also *induce* additional ill-conditioning (which would have been avoided by using a stable algorithm). Since the straight line is a special case of a polynomial the above causes can also apply, although the first cause does not because in this case the Chebyshev polynomials and the monomials are identical.

For a chosen value of $m$, a basic data set $\{(x_i, y_i)\}_1^m$ was devised by generating values $y_i$ lying exactly on a straight line at $m$ equispaced points $x_i$ in [-1, 1]. Null-space methods were then used to perturb these values to simulate random noise. The actual test detailed below applied to $m = 41$ but, to illustrate the null-space approach, the case $m = 3$ is considered first.

The $x$-values in the case $m = 3$ are -1, 0 and +1. The corresponding Jacobian matrix (see Section 4.1) is

$$J = \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix},$$

and, as can easily be verified by forming $J^{\mathrm{T}}N$, the corresponding null-space matrix is

$$N = \frac{1}{\sqrt{6}} \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}.$$

Now, for any specific given straight line, generate the three $y$-values corresponding to these values of $x$. If the line is

$$y = 5 + 2x, \tag{7}$$

the $y$-values would be 3, 5 and 7. From the theory, any vector that can be written as a linear combination of the columns of $N$ can be added to the vector of $y$-values, and the least-squares straight line for these perturbed values would be identical to the original line (7). In this simple case the null matrix is a vector, so any *multiple*, $\lambda\sqrt{6}$ say, of the null-space vector can be added to the $y$ vector to give

$$y_{\mathrm{new}} = \begin{bmatrix} 3+\lambda \\ 5-2\lambda \\ 7+\lambda \end{bmatrix}.$$

A small value of $\lambda$ can be used to simulate a small-residual problem and a large value a large-residual problem. The significance of this statement is that some software can perform less well for large-residual problems and graded data sets spanning a wide range of $\lambda$ can be helpful in identifying the point at which, for practical purposes, accuracy starts to be lost. It is to be noted that for nonlinear least-squares problems, for which iterative methods of solution, such as Gauss-Newton and its variants [5], are employed, the convergence rates of these methods are affected by the size of the residuals. Thus the performance (in terms of execution time, accuracy of solution, ability to converge to the required solution, etc.) of such algorithms with respect to this "size" can be examined. For straight-line regression, a more important effect is the data normalisation. In this regard, the performance of test software to the choice of origin for the $x$-values can be investigated. If a constant is added to the $x$-values, the solution mathematically will be trivially changed (Section 2.3): the gradient will be unaffected, but the intercept term, if the parametrisation (1) is used, will change. The change in the intercept value is proportional to the constant translation of the $x$-values. The residuals are of course unaffected.

Two items of straight-line regression software were applied to a data set containing $m = 41$ points and performance measure $P$ based on (3) applied to the residuals of the fitted function determined in each case. The resulting performance profiles ($P$ against degree of difficulty $K$ equal to the distance of the data set from the origin) are shown in Figure 2. The generally-lower line was obtained by joining the points representing the results for software based on a stable algorithm, A, which uses a normalised variable, and the resulting overdetermined linear system solved using QR decomposition [6, p239] (a stable algorithm for linear-least-squares problems). The generally-higher line corresponds to values obtained from the use of a less stable algorithm, B, which uses the untransformed variable and forms and solves the normal equations.

Note that Algorithm A performed near-optimally, the software based on Algorithm B steadily losing accuracy as $K$ increased.

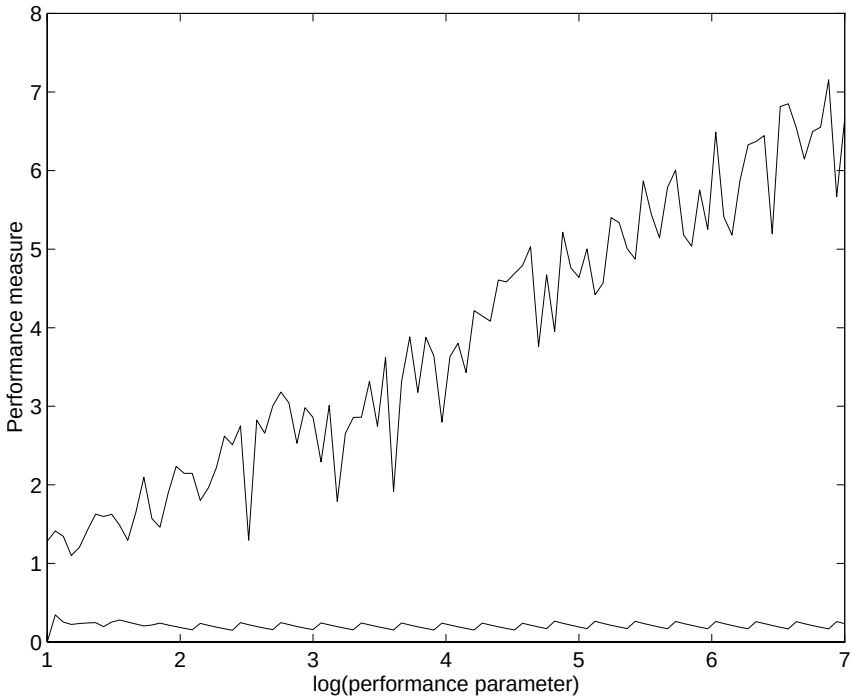


Figure 2: Performance profile for two items of straight-line-regression software when used to fit 41 equispaced data points. Generally-lower line: Algorithm A (normalised data, QR decomposition); generally-upper line: Algorithm B (unnormalised data, formation and solution of the normal equations).

## 5.3 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 [16]. 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 normalized eigenvectors associated with the $l$ most significant eigenvalues $\lambda_i$ of the normal matrix $H = X^T X$ [17].

There are two predominant algorithms for solving the PCA problem. One involves forming $H = X^{\mathrm{T}}X$ explicitly and employing eigendecomposition software: $H = VDV^{\mathrm{T}}$ [6, 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) [6, p69] of $X$: $X = USV^{\mathrm{T}}$, 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 $X = USV^{\mathrm{T}}$ implies that $H = X^{\mathrm{T}}X = VS^2V^{\mathrm{T}}$, 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 [2]. Consider a $9 \times 6$ matrix $X$ which is

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

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

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

$X$ 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 $2 \times 10^{-12}$. This result is close to best possible for the computational precision $\eta$ and the degree of difficulty $K$. For eigendecomposition-based software the $V$-matrix computed differed in 2-norm from the true $V$ by $2 \times 10^{-7}$. 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 $9 \times 6$ matrices, i.e., a sequence whose condition numbers successively increase, a performance profile as in the earlier examples 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. Concluding remarks

This paper has indicated the nature of some of the work carried out at the National Physical Laboratory on the use of reference data sets for testing software used in scientific applications. Emphasis has been placed on the use of graded reference data sets to produce performance profiles of test software. Examples of their use and benefits have been provided.

Further work in this area will be carried out under the Software Support for Metrology programme of the UK Department of Trade and Industry (DTI). This programme includes the use of reference data sets to test key mathematical functions embodied in widely-used spreadsheet systems and other software packages.

Much of the technical work described here was carried out as part of the Valid Analytical Measurement programme of the DTI and as a supplementary project supported by the National Measurement System Policy Unit of the DTI.

Bernard Butler provided many valuable comments on the draft of this paper.

## References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

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

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

[8] ISO. ISO/DIS 10360-6. Geometrical product specifications (GPS) – acceptance test and reverification test for coordinate measuring machines (CMM). Part 6: Computation of Gaussian associated features, 1998. International Standards Organisation proposed draft standard.

[9] 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.

[10] 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.

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

[12] R. Drieschner, B. Bittner, R. Elligsen, and F. Waeldele. Testing coordinate measuring machine algorithms. Phase II.Technical Report BCR EUR 13417 EN, Commission of the European Communities, 1991.

[13] 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.

[14] 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.

[15] C. W. Clenshaw. *Mathematical Tables Volume 5. Chebyshev Series for Mathematical Functions.* Her Majesty's Stationery Office, 1962.

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

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