NPL Report CISE 26/99

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

TESTING SPREADSHEETS AND OTHER PACKAGES USED IN METROLOGY

A CASE STUDY

BY

H R COOK, M G COX, M P DAINTON

and P M HARRIS

September 1999

Testing Spreadsheets and Other Packages Used in Metrology
A Case Study

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

September 1999

ABSTRACT

A case study that illustrates the application of a general methodology for testing the numerical accuracy of scientific software and showing its "fitness for purpose" within a given application is described. Each stage of the methodology, from the specification of the problem addressed by the test software through to the presentation and interpretation of the results of the testing, is outlined. The methodology is then applied to the test problem: fit a single Gaussian peak to measurement data.

The methodology can be used by *developers* of metrology software to establish the quality of the software they write. The methodology is equally applicable to the testing of software provided with proprietary software packages by *users* of such software.

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

ISSN 1361-407X

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

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

Contents

1. Introduction

A general methodology for evaluating the numerical accuracy of the results produced by scientific software is described in [1, 2]. The basis of the approach is the design and use of reference data sets and corresponding reference results to undertake black-box testing of the software to be tested. The reference data sets and results are generated in a manner consistent with the functional specification of the test software, and the results returned by the test software for the reference data are then compared objectively with the reference results using quality metrics or performance measures. Finally, the performance measures are interpreted in order to decide whether the test software meets requirements and is fit for its intended purpose.

The purpose of this report is to describe a case study that illustrates the application of this methodology to test software developed to solve a typical problem arising in metrology. Each stage of the process, from the specification of the problem through to the presentation and interpretation of test results, is described in generic terms in [2]. The stages are illustrated here for the particular problem of fitting a single Gaussian peak to measurement data. Further examples of the application of the methodology to testing functions from the Excel spreadsheet package are presented in [3].

The case study is intended to provide an example of how the general testing methodology may be applied both by *developers* and *users* of metrology software. For developers of metrology software, there is a growing need to demonstrate to users that the software is fit for purpose and especially that the results it produces are correct, to within a specified accuracy, for the problems purportedly solved. For users of proprietary software packages in metrology applications, there is likewise a requirement to demonstrate the fitness for purpose of the software. The testing methodology helps to address these requirements by providing developers and users of metrology software with a procedure that yields quantitative statements about the performance of the software.

The report is organised as follows. In Section 2 we provide a summary of the testing methodology and give an overview of each stage of the approach. In Section 3 the application of each of these stages to the test problem “Fit a Gaussian peak to measurement data” is presented. Our conclusions are given in Section 4.

2. Methodology

The approach to testing comprises the following six stages:

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

These stages are summarised briefly here: for full details, see [2].

Specification of the test software

For black-box testing of the test software to be meaningful, it is necessary to have a full and unambiguous specification of the task carried out by the software. If the specification is inconsistent with the task carried out by the software, testing in accordance with the

specification would likely yield the conclusion that the software was deficient, when in fact it might be performing its task acceptably.

The specification may be given in mathematical or non-mathematical terms, but in either case enough information must be available to avoid ambiguity. One way to provide the specification is to define for the test software:

a) its inputs,

b) its outputs, and

c) the functional relationship or model that relates the inputs to the outputs.

Implementation of the test software

For the software *user*, the test software typically takes the form of a given software function, perhaps included as an in-built function within a commercial spreadsheet or other software package. Alternatively, for the software developer, the test software is typically implemented using appropriately chosen (lower-level) in-built functions. For the viewpoint of its testing, however, no distinction is necessary, since in either case the test software is regarded as a "black-box".

In addition to the test software itself, software may usually be required for

a) generating reference data sets and corresponding reference results, and

b) applying the test software to the reference data sets and subsequently evaluating the quality metrics and performance measures discussed below.

An alternative to a) is to obtain reference data sets and corresponding reference results from a reputable source. An issue related to b) is the management of the interface between this software and the test software. For a developer of the test software, the interface can usually be made quite seamless. For a user of the test software, the interface may be more cumbersome unless the design of the software facilitates the interface. In addition, it must be possible to pass input data to the test software and extract the appropriate output results to a numerical precision sufficient to enable a meaningful comparison between test and reference results.

Specification of reference data sets

Performance parameters that describe the scope of admissible inputs to the test software are identified. Sequences of data sets may then be defined for which, within a given sequence, different data sets correspond to different choices of a particular performance parameter (or parameters). The performance of the test software is investigated for such sequences of data sets. Where a performance parameter can be interpreted as controlling the condition or "degree of difficulty" of the data set, the sequence of data sets becomes a graded sequence in the sense that each data set in the sequence represents a problem that is more difficult than the previous member of the sequence. Use of such graded sequences of data sets can help to identify cases where the test software is based on a poor choice of mathematical algorithm.

In addition, the reference data sets may be required to reflect as far as possible data sets that would be obtained and used in practical measurement. The specification of the reference data sets should identify such requirements, for example, by identifying performance parameters that capture these requirements and constraining these parameters accordingly.

Specification of performance measures and testing requirements

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

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

a) the difference between the test and reference results, an *absolute* measure of departure,

b) the number of figures of agreement, a *relative* measure of departure, and

c) a *performance measure* that accounts for the factors including the computational precision of the arithmetic used to generate the test and reference results and the conditioning of the problem.

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

1. memory requirements,

2. and execution time.

Furthermore, in situations where a number of test implementations are available, as in the example reported here, metrics may be used that measure the difference between the test results returned by these various implementations as well as to report their absolute performance compared with reference results.

Generation of reference data and reference results

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

a) Start with a reference data set and apply reference software to it to produce the corresponding reference results.

b) Start with reference results and apply a data generator to them to produce a corresponding reference data set.

The approach generally adopted within the methodology described in [1, 2] is to use a data generator to construct reference data sets to have known solutions, i.e., solutions specified *a priori*. Data generators can generally be implemented for problems for which a mathematical characterisation of a solution to the problem exists, and this applies to a wide range of calculations used in metrology.

For example, given a regression model $\mathbf{f}(\mathbf{a})$ in terms of regression parameters $\mathbf{a}$ and observations $\mathbf{y}$, the solution to the regression problem of finding the least-squares best-fit model for the observations is characterised by the condition

$$J^{\mathrm{T}}\mathbf{e} = 0, \tag{1}$$

where $\mathbf{e}$ is the vector of residual errors at the solution,

$$\mathbf{e} = \mathbf{y} - \mathbf{f}(\mathbf{a}),$$

and J is the Jacobian matrix for the residual errors with respect to the regression parameters $\mathbf{a}$, i.e.,

$$J_{ij} = \frac{\partial e_i}{\partial a_j}.$$

Based on the condition (1), a procedure for generating a reference data set $\mathbf{y}$ given the regression model $\mathbf{f}$ and parameters $\mathbf{a}$ then takes the following form [2]:

I. Evaluate the observation vector $\mathbf{y}_0$ given by $\mathbf{y}_0 = \mathbf{f}(\mathbf{a})$.

II. Evaluate the Jacobian matrix J.

III. Form null space matrix N for J^{T}.

IV. Form residual vector **e** given by **e** = N**u**, where the elements of **u** are chosen to be random numbers.

V. Form observation vector **y** given by **y** = $\mathbf{y}_0$ + **e**.

In the case that a vector $\mathbf{e}_0$ of "target" residuals is specified for the reference data set, step IV is replaced by [2]:

IV. Form residual vector **e** given by **e** = N**u**, where **u** = $N^{\mathrm{T}}\mathbf{e}_0$.

The resulting reference data set will have a known solution **a** characterised by a known residual vector **e** that is close (as measured by the least-squares or l_2-norm) to the vector $\mathbf{e}_0$ of "target" residuals.

Presentation and interpretation of calculated performance measures

Having applied the test software to a reference data set to give a test result, the test result is compared with the reference result corresponding to the data set by computing one or more quality metrics. The quality metrics are presented as functions of a performance parameter (or parameters) either

a) in tabular form, and/or

b) as a graph by plotting it against the performance parameter, i.e., as a performance profile.

It also convenient to present summary statistics, either numerically or graphically, including the arithmetic mean, standard deviation, minimum and maximum, of the computed metrics.

The values of the computed quality metrics are compared against the requirements on the test software expressed in terms of these metrics. A judgement about the quality of the test software, and its suitability for a particular application, can then be made in an objective manner.

3. Gaussian Peak Fitting

In this section we present the particularisation of each of the stages of the testing methodology outlined in Section 2 to the test problem "Fit a Gaussian peak to measurement data".

3.1 Specification of test software

For the case study considered here, the following statement may be regarded as sufficient:

> "The test software computes the least-squares best-fit Gaussian peak to data with measurement error in the values of the dependent variable only."

Alternatively, we may define the task carried out by the test software in terms of its inputs, outputs and input/output model as follows:

a) *Inputs*: Abscissa values x_i, i = 1, …, m, and corresponding ordinate values y_i, i = 1, …, m.

b) *Outputs*: Parameter values A, $\bar{x}$ and s in the model for a Gaussian peak

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

and residual values

$$e_i,\ i = 1, \ldots, m, \text{ where } e_i = y_i - y(x_i). \tag{3}$$

c) *Input/output model*: The outputs are determined from the inputs by solving the following (non-linear) least-squares problem:

$$\min_{A,\bar{x},s} \sum_{i=1}^{m} \{y_i - y(x_i)\}^2, \tag{4}$$

and evaluating the residual values using (4).

3.2 Implementation of test software

Suppose, for a given x_0, we re-parametrise the model (2) in the form

$$y(x) = \exp\{a_1 + a_2(x - x_0) + a_3(x - x_0)^2\}, \tag{5}$$

where

$$\begin{aligned} A &= \exp\left(a_1 - \frac{a_2^2}{4a_3}\right), \\ s &= \sqrt{\frac{-1}{2a_3}}, \\ \bar{x} &= x_0 - \frac{a_2}{2a_3}, \end{aligned} \tag{6}$$

gives the relationship between the two parametrisations. Then, test software for the problem of fitting a Gaussian peak to measurement data might implement the following algorithm:

Step 1:

Taking natural logarithms of (5), we obtain

$$\ln y(x) = a_1 + a_2(x - x_0) + a_3(x - x_0)^2,$$

and so initial estimates for a_1, a_2, a_3 are obtained by solving the *linear* least-squares problem:

$$\min_{a_1,a_2,a_3} \sum_{\{i:y_i>0\}} y_i^2 \{\ln y_i - \ln y(x_i)\}^2. \tag{7}$$

The inclusion of the factor y_i^2 in the above summation gives (partial) compensation for bias introduced by the (natural logarithm) transformation and its effect on the data errors. Furthermore, in order to handle the case $y_i \le 0$ for which we may not take (real-valued) natural logarithms, the summation is restricted to those data points for which $y_i > 0$.

Step 2:

Apply the Gauss-Newton algorithm [4] to determine final estimates for a_1, a_2, a_3. The Gauss-Newton algorithm is an iterative algorithm where at the kth iteration we solve the *linear* least-squares problem:

$$J_k \delta \mathbf{a}_k = -\mathbf{e}_k. \tag{8}$$

Here, $\mathbf{e}_k$ is a vector containing the residual values for the current estimates $\mathbf{a}_k$ of the solution parameters, J_k is the Jacobian matrix, i.e., the matrix of partial derivatives

$$\frac{\partial e_i}{\partial a_j}$$

evaluated at $\mathbf{a}_k$ and $\delta\mathbf{a}_k$ contains the Gauss-Newton step from which a new estimate $\mathbf{a}_{k+1}$ of the solution parameters is calculated:

$$\boldsymbol{a}_{k+1} = \boldsymbol{a}_k + \delta\boldsymbol{a}_k .$$

The iterative procedure is terminated when either

a) a specified maximum of iterations has been performed, or

b) the condition

$$\boldsymbol{e}_{k+1}^{\mathrm{T}}\boldsymbol{e}_{k+1} \geq \boldsymbol{e}_k^{\mathrm{T}}\boldsymbol{e}_k$$

is satisfied, i.e., there is no decrease in the value of the objective function.

If case b) applies, the algorithm has converged, and estimates of the solution parameters are returned. If case a) applies, estimates of the solution parameters are not returned.

Step 3:

Use equation (6) to evaluate parameter values A, $\bar{x}$ and s corresponding to the final estimates for a_1, a_2, a_3, and equations (2) and (3) to evaluate the residual values e_i, $i = 1, \ldots, m$.

Software implementing the steps described above has been developed in MATLAB [4]. The software uses MATLAB's in-built "\" (backslash) operator to solve the linear least-squares problems arising in Steps 1 and 2. The intention here is that the algorithm implemented is representative of software that might be used in practice: the software is a faithful implementation of a Gauss-Newton algorithm but does not include features, such as a line search, that might improve its performance.

In this case study, two implementations of the test software are considered:

Test Software A: x_0 is set equal to zero.

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

These choices of the constant x_0 correspond to two different parametrisations of the Gaussian peak model (5).

3.3 Specification of reference data sets

Performance parameters for the Gaussian peak fitting problem are chosen as follows:

a) peak location $\bar{x}$;

b) peak width s;

c) peak height A;

d) data noise σ;

e) the number m of abscissa values x_i;

f) data location x_0, i.e., the midpoint of the span of the x_i values;

g) data width w, i.e., the semi-range of the x_i values, given by $\frac{1}{2}(\max\{x_i\} - \min\{x_i\})$.

The first three of these are the parameters used to define the model (2) in the problem specification given in Section 3.1, and therefore control the range of solutions returned by the test software. The remainder of the performance parameters relate to properties of the input data provided to the test software.

In Table 1 we give nominal, minimum and maximum values for the performance parameters listed above. The values given in this table control the range of reference data sets used to test the software. If Gaussian peak-fitting software were to be used in a particular environment or application, these controlling values should contain, or if necessary be replaced by, those

relevant for that environment. In Figure 1 we illustrate the Gaussian peak and an input data set corresponding to the nominal values for the performance parameters given in Table 1. The data in this figure is calculated by evaluating the model for the Gaussian peak (defined by the nominal values of A, $\bar{x}$ and s) at x-values (defined by the nominal values of m, x_0 and w) and adding random measurement errors sampled from a Gaussian probability distribution (with mean zero and standard deviation equal to the nominal value of σ).

Seven sequences of reference data sets are used to test the software. These sequences are defined by the performance parameters as follows. For the first sequence, $\bar{x} = x_0 \in [996, 1004]$, with all other performance parameters set equal to their nominal values given in Table 1. For the remaining sequences, each of the performance parameters s, A, σ, m, x_0 and w is varied between the minimum and maximum values given in Table 1 with all other performance parameters set equal to their nominal values given in Table 1.

It should be noted that the testing described here does not involve data sets for which more than one of the performance parameters takes extreme values. The generation of individual reference data sets and corresponding reference results is described in Section 3.5.

Performance parameter	*Nominal value*	*Minimum*	*Maximum*
$\bar{x}$	1000	$\bar{x} - 4s = 996$	$\bar{x} + 4s = 1004$
s	1	$\frac{s}{10} = 0.1$	$10s = 10$
A	$\frac{1}{\sqrt{2\pi}}$	$\frac{A}{10} = \frac{1}{10\sqrt{2\pi}}$	$10A = \frac{10}{\sqrt{2\pi}}$
σ	$\frac{1}{20\sqrt{2\pi}}$	$\frac{\sigma}{50} = \frac{1}{1000\sqrt{2\pi}}$	$4\sigma = \frac{1}{5\sqrt{2\pi}}$
m	100	5	300
x_0	1000	$x_0 - 2s = 998$	$x_0 + 2s = 1002$
w	1	$\frac{w}{20} = \frac{1}{5}$	$20w = 20$

Table 1: Nominal, minimum and maximum values for the performance parameters chosen for the example of fitting a Gaussian peak to measurement data.

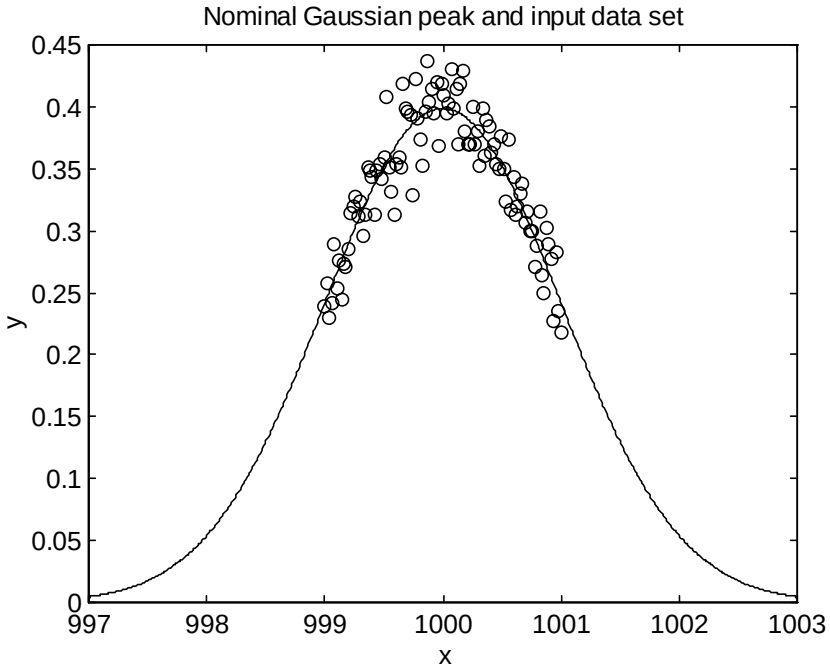

Figure 1: Gaussian peak and (typical) input data set defined by the nominal values of the performance parameters chosen for the example of fitting a Gaussian peak to measurement data.

3.4 Specification of performance measures and testing requirements

We measure the *absolute* performance of each test software implementation for fitting a Gaussian peak to data using two quality metrics.

The first of these is a performance measure $P(\mathbf{e})$ for the departure of the residual errors returned by the test software from the reference residuals:

$$P(\boldsymbol{e}) = \log_{10}\left(1 + \frac{\|\boldsymbol{e}_t - \boldsymbol{e}_r\|}{\|\boldsymbol{y}\|\eta}\right),$$

where $\mathbf{e}_t$ is the vector of residual errors returned by the test software, $\mathbf{e}_r$ is the vector of reference residuals, $\mathbf{y}$ is the vector of reference data values, and η is the computational precision of the arithmetic used to generate the test results. $P(\mathbf{e})$ indicates the number of figures of accuracy *lost* by the software over and above what software based on an optimally stable algorithm would produce: see [2].

The second quality metric is the number N of iterations taken by the test software to converge to a solution (with $N = \infty$ denoting no convergence within the maximum number of iterations allowed).

We measure the relative performance of the two test software implementations using the performance measures $P(A)$, $P(s)$ and $P(\bar{x})$ where

$$P(u) = \log_{10}\left(\frac{|u_{t,A} - u_r|}{|u_{t,B} - u_r|}\right),$$

with u replaced by A, s and $\bar{x}$, and where $u_{t,A}$, $u_{t,B}$ are the test results returned by Test Software A and B, respectively, and u_r is the reference result. (Here, to ensure that $P(u)$ always takes a finite value, the difference between a test result and the reference result is replaced by $\eta|u_r|$ whenever that difference is computed as zero.) $P(u)$ indicates the number of figures of accuracy *lost* by Test Software A compared with Test Software B. Because $P(u)$ does not reflect the

condition of the problem of computing the parameter u or the computational precision of the arithmetic used to generate the test results, it does not provide an absolute measure of performance. However, it does enable the *ranking* of test software implementations in the sense that $P(u) > 0$ indicates that Test Software B returns a more accurate value for u than Test Software A, and $P(u) < 0$ indicates that Test Software A returns a more accurate value than Test Software B.

The test requirements for the test software are expressed as follows:

a) $P(\mathbf{e}) < 8$,

b) $N < 20$, and

c) in cases where both a) and b) are satisfied, to identify the test software implementation that returns values for A, s and $\bar{x}$ to the highest accuracy over the range of problems and data sets defined by the performance parameters and their values given in Table 1.

3.5 Generation of reference data and reference results

For the example of fitting a Gaussian peak to data, a single reference pair is generated in the following way (Section 2 and [2]):

a) The values of the performance parameters m, x_0 and w are used to define abscissa values x_i, $i = 1, \ldots, m$. In addition, the abscissa values are also required to be equispaced.

b) The values of the performance parameters A, s, and $\bar{x}$, and the abscissa values x_i are used to define ordinate values p_i, $i = 1, \ldots, m$, that lie exactly on the Gaussian peak defined by A, s and $\bar{x}$.

c) The value of the performance parameter σ is used to define a set of target residual errors v_i, $i = 1, \ldots, m$, that are samples from a Gaussian distribution with mean zero and standard deviation σ. In Figure 2 an example set of target residual errors v_i is plotted against the abscissa values x_i.

d) The null-space method is used to determine residual errors e_i, $i = 1, \ldots, m$, that are close to the target residual errors v_i and satisfy the characterisation (1).

e) Data ordinate values y_i, $i = 1, \ldots, m$, are defined by $y_i = p_i + e_i$.

The abscissa values x_i and ordinate values y_i define a reference data set for which the reference results are A, s, $\bar{x}$ with residual errors e_i. A reference data set generated in this way together with the known underlying best-fit Gaussian peak is shown in Figure 3. The corresponding reference residual errors e_i are plotted in Figure 4. In Figures 2–4 the performance parameters have been set equal to their nominal values. Although the data sets shown in Figures 1 and 3 appear to be the same, they are not identical (and similarly for the residual errors shown in Figures 2 and 4). This shows that it is possible to use the null-space approach to generate a reference data set with residual errors that mimic very closely the target residual errors.

The procedure described above is repeated (many times) for the same values of the performance parameters but different choices of target residual errors v_i to obtain reference data sets with the same reference results A, s and $\bar{x}$ but different reference residuals e_i. The procedure is also repeated (many times) with different values of the performance parameters to obtain sequences of reference data sets.

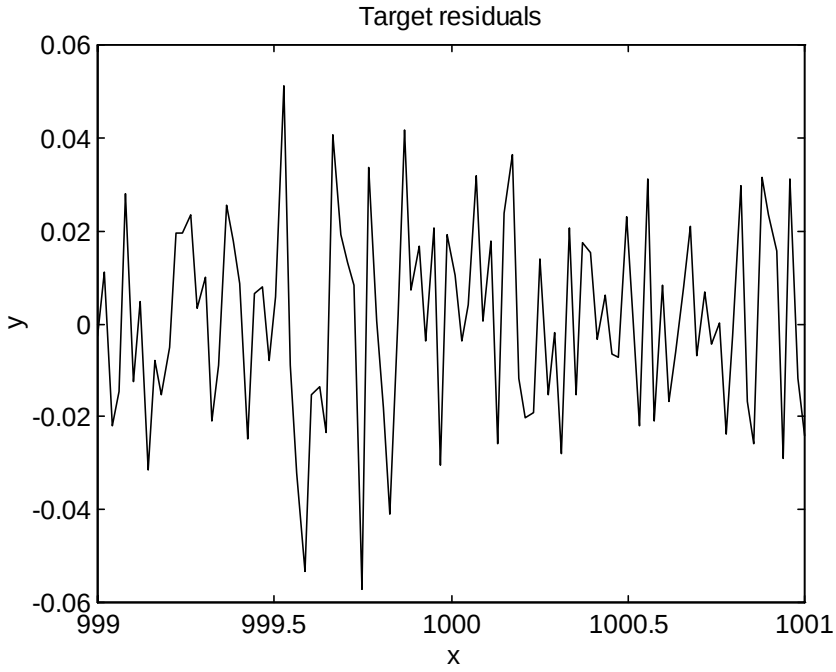

Figure 2: Target residual errors obtained by sampling a Gaussian probability distribution with mean zero and standard deviation σ.

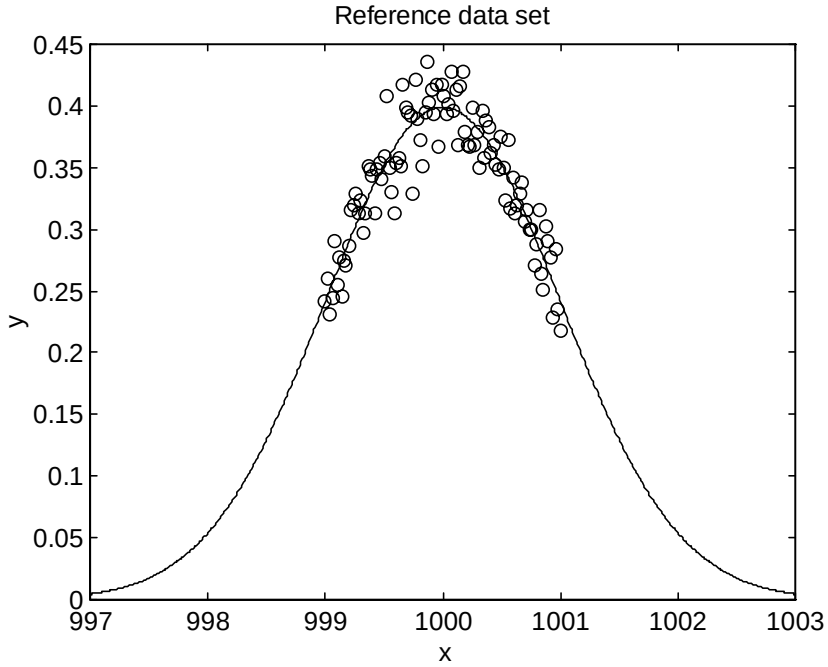

Figure 3: Reference data generated using the "null-space" method. The *known* Gaussian peak that is the least-squares best-fit to the reference data is also shown.

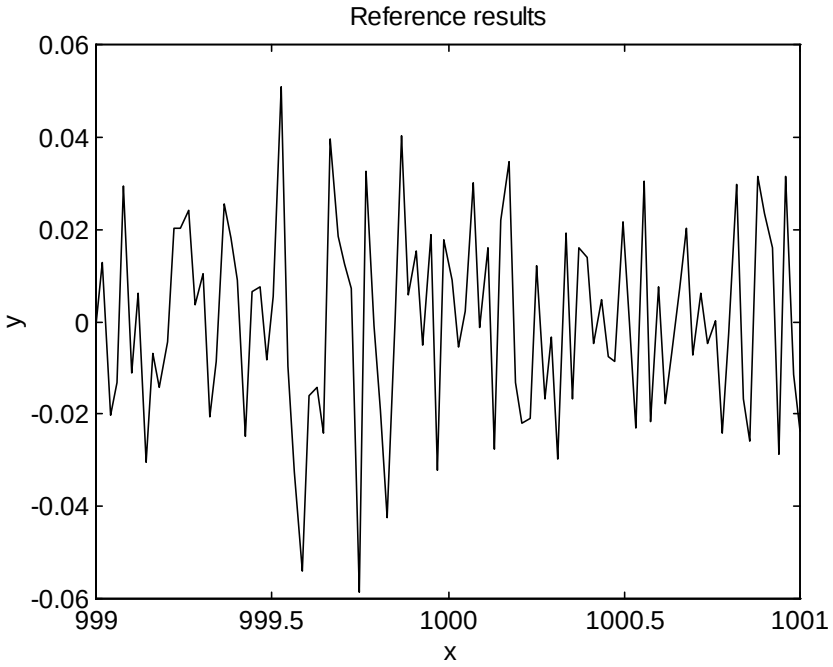

Figure 4: Reference residuals for the reference data set shown in Figure 3. The reference residuals are chosen to be close to the target residuals shown in Figure 2 in order to simulate random measurement error.

3.6 Presentation and interpretation of calculated performance measures

In Figures 5–11 we present graphs of $P(\mathbf{e})$ and N for the two test software implementations and the seven sequences of reference data sets. For each value of the performance parameter, 100 reference data sets are constructed and the minimum, mean and maximum of the two performance measures for these data sets calculated. The graphs show the minimum, mean and maximum values joined by straight-lines: results for Test Software A are shown using solid lines, and those for Test Software B using broken lines. Similarly, Figures 12–18 show graphs of $P(A)$, $P(s)$ and $P(\bar{x})$ for the two test software implementations and the seven sequences of reference data sets computed in the same way.

We make the following observations about the test software.

The performance of the two test software implementations is insensitive to the peak and data locations where these are equal (Figure 5). However, the performance degrades as

- the peak width is reduced (Figure 6),
- the peak height is reduced (Figure 7),
- the data noise is increased (Figure 8),
- the number of data points is reduced (Figure 9),
- the difference between the data location and the peak position is increased (Figure 10), and
- the data width is increased (Figure 11).

Generally, we expect the software to perform worse when the peak is less well defined by the data. This happens when there is a greater amount of data noise or when there are fewer points defining the peak. Since the data is discrete and equispaced in its independent variable x, the number of points defining the peak is directly controlled by the peak width, peak height, number of data points, data location/peak position difference and data width. Consequently, we can expect the results indicated above.

In addition, the data noise affects the quality of the initial estimates of the model parameters and, therefore, the number of iterations required by the Gauss-Newton algorithm to converge to a solution. (The Gauss-Newton algorithm is known also to have a faster rate of convergence for small residual problems.)

Using the measure $P(\mathbf{e})$ of numerical accuracy, and by considering the "mean" curves shown in the figures, we observe that on average Test Software B delivers consistently a more accurate result than Test Software A. However, there are examples of data sets within the range of admissible inputs for which Test Software A can deliver a result that is more accurate than that returned by Test Software B. An example of this is for large peak widths for which the "minimum" curve for Test Software A is below that of the "maximum" curve for Test Software B (Figure 6).

Using the measure N of performance, Test Software B requires on average more iterations than Test Software A to converge to a solution.

Recall (Section 3.4) that the test requirements for the test software are expressed as follows:

a) $P(\mathbf{e}) < 8$,

b) $N < 20$, and

c) in cases where both a) and b) are satisfied, to identify the test software implementation that returns values for A, s and $\bar{x}$ to the highest accuracy over the range of problems and data sets defined by the performance parameters and their values given in Table 1.

Test Software B always returns a result that satisfies the test requirement a) given above, whereas there are examples of data sets within the range of admissible inputs for which Test Software A fails to meet this requirement. Examples of this include large values for the data noise (Figure 8), and large values for the difference between the data location and the peak position (Figure 10).

Both test software implementations return results that satisfy the test requirement b) given above. For data sets with a large data width, however, some results are obtained which are close to failing the test requirement.

Test Software B returns estimates for the model parameters A, s and $\bar{x}$ that are, on average, more accurate than those returned by Test Software A.

We conclude that Test Software B meets the specified test requirements whereas Test Software A does not.

As far as the accuracy requirement is concerned, we might expect this result. Recall that the test software implementations differ in the way that the parameter x_0 is chosen, with x_0 set equal to zero for Test Software A and equal to the arithmetic mean of the data x_i values for Test Software B. Centering of the data, as is done by Test Software B, is generally beneficial from a numerical point of view. The transformation is used in other calculations, e.g., the sample standard deviation [2] for which it is shown that the condition of the transformed problem is optimal. A similar beneficial effect can be expected for the peak fitting example considered here.

Finally, we note that the test software implementations considered here are both "correct" in the sense that they are each capable of delivering sound results, with the accuracy of these results improving as the computational precision η is decreased (i.e., improved). However, for software implementing less sophisticated algorithms, e.g., take logarithms and fit a weighted quadratic polynomial (Step 1 of the algorithm described in Section 3.2), that solve approximations to the peak fitting problem, decreasing η cannot help in producing a better solution to the true problem.

4. Conclusion

In this report we have described a case study that illustrates the application of a general methodology for testing the numerical accuracy of scientific software and showing its "fitness-for-purpose" within a given application. Each stage of the methodology, from the specification of the problem solved by the test software through to the presentation and interpretation of test results, has been outlined. The stages have then been illustrated for a test problem that is representative of metrology applications: fit a single Gaussian peak to measurement data. We have shown how the methodology can be used by *developers* of metrology software to establish the quality of the software they write. The approach is equally applicable to the testing of software provided as part of proprietary software packages, for example spreadsheets, by *users* of such software. Examples of the application of the methodology within the latter context are provided in [3].

5. Acknowledgement

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

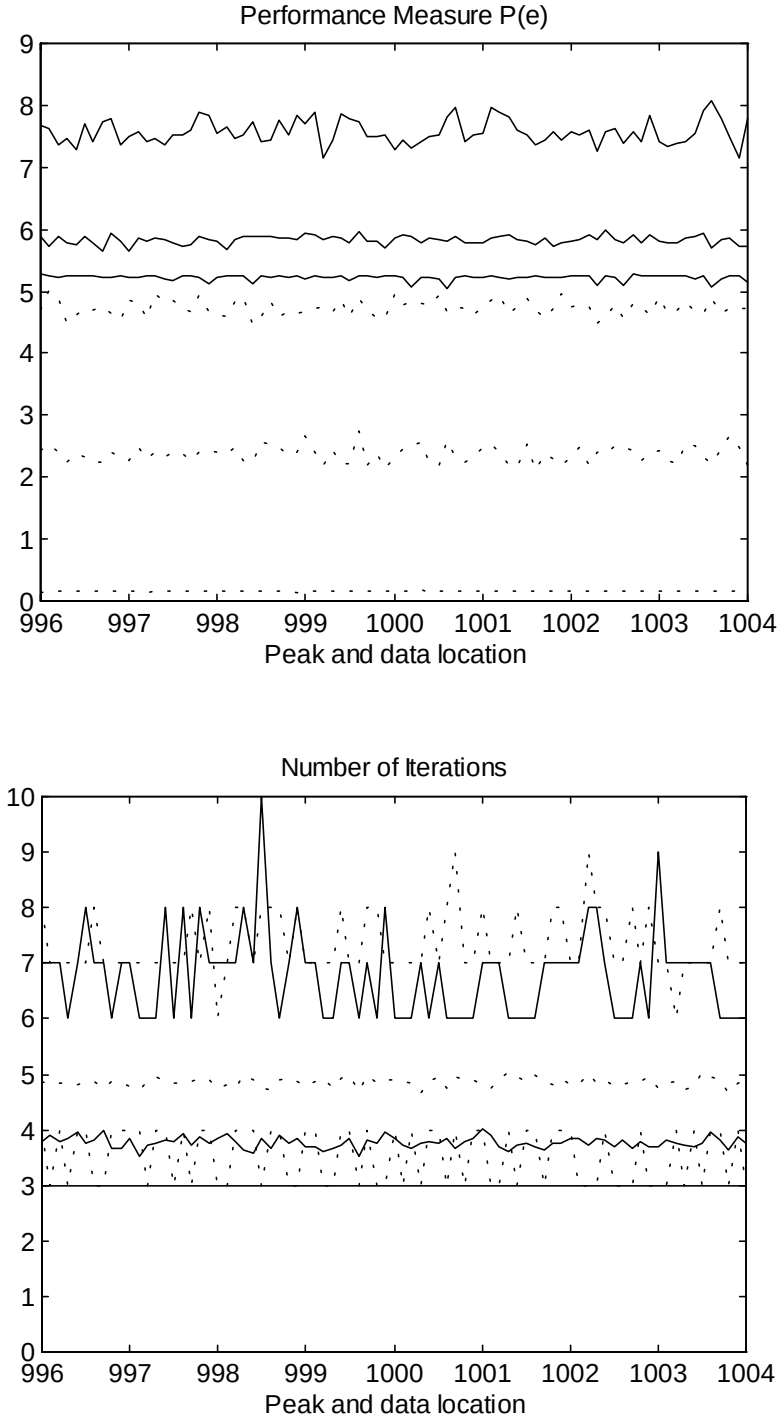

Figure 5: Values of the quality metrics *P*(**e**) and *N* for the two test software implementations as a function of peak and data location. Results (minimum, mean and maximum) for Test Software A are shown using solid lines, and those for Test Software B using broken lines.

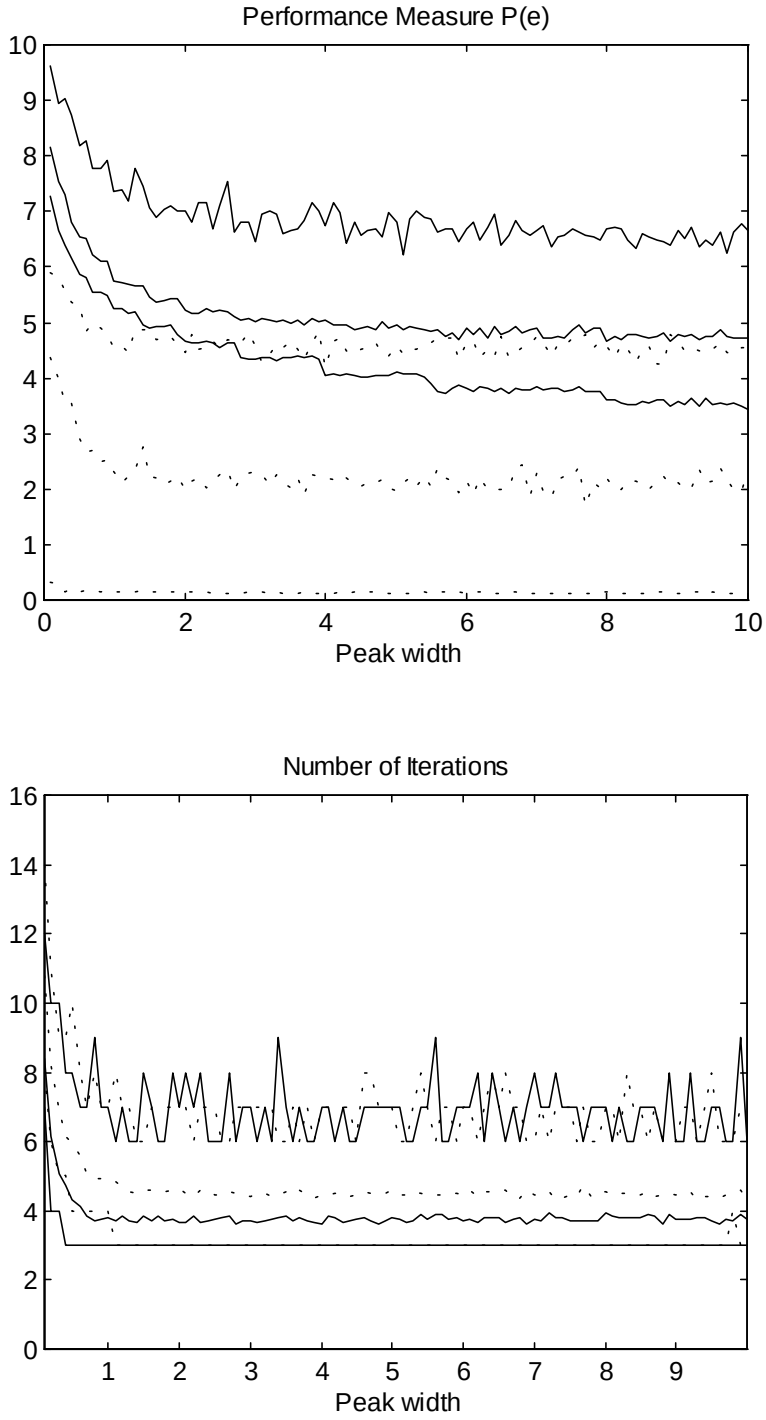

Figure 6: Values of the quality metrics $P(\mathbf{e})$ and N for the two test software implementations as a function of peak width. Results (minimum, mean and maximum) for Test Software A are shown using solid lines, and those for Test Software B using broken lines. (In the second of these figures, the minimum curves for the two implementations coincide for a large part of the range of peak width.)

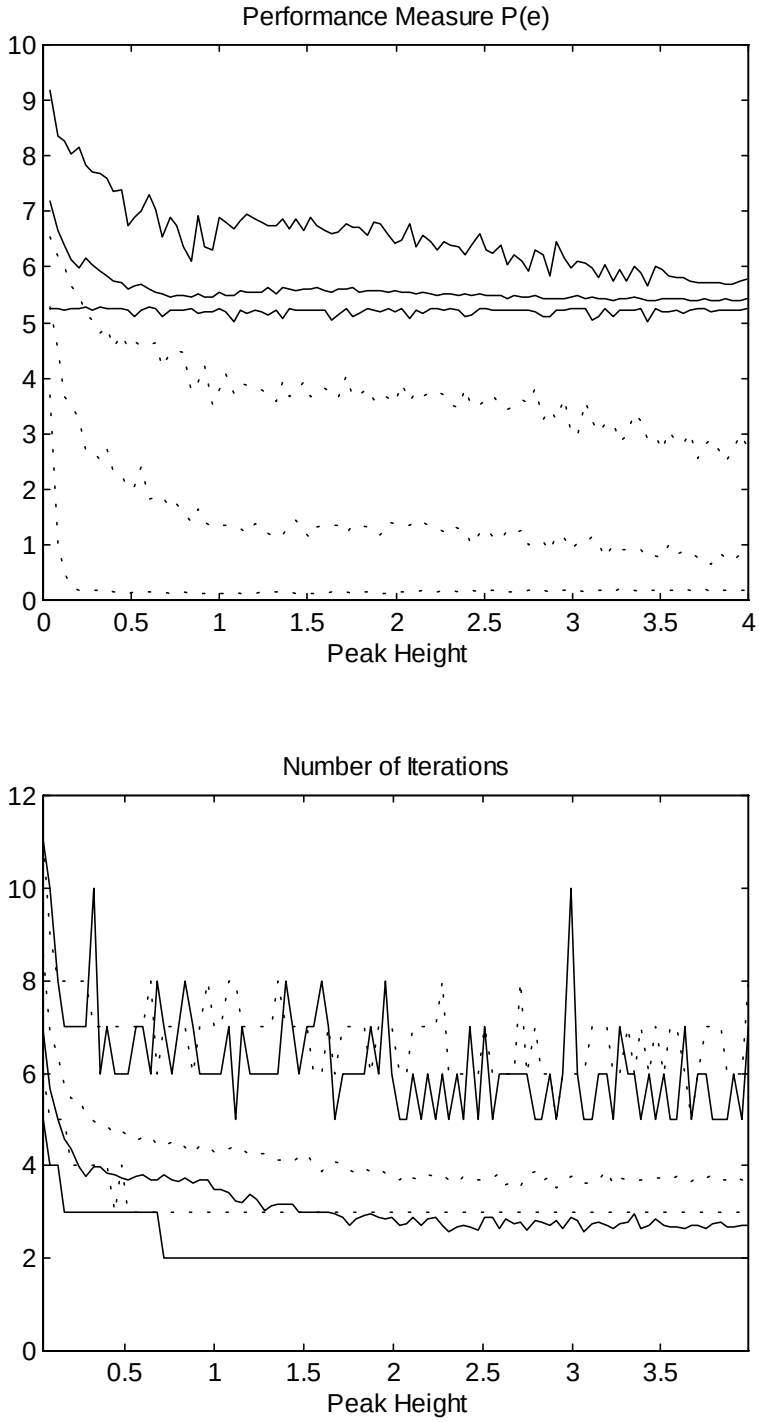

Figure 7: Values of the quality metrics $P(\mathbf{e})$ and N for the two test software implementations as a function of peak height. Results (minimum, mean and maximum) for Test Software A are shown using solid lines, and those for Test Software B using broken lines.

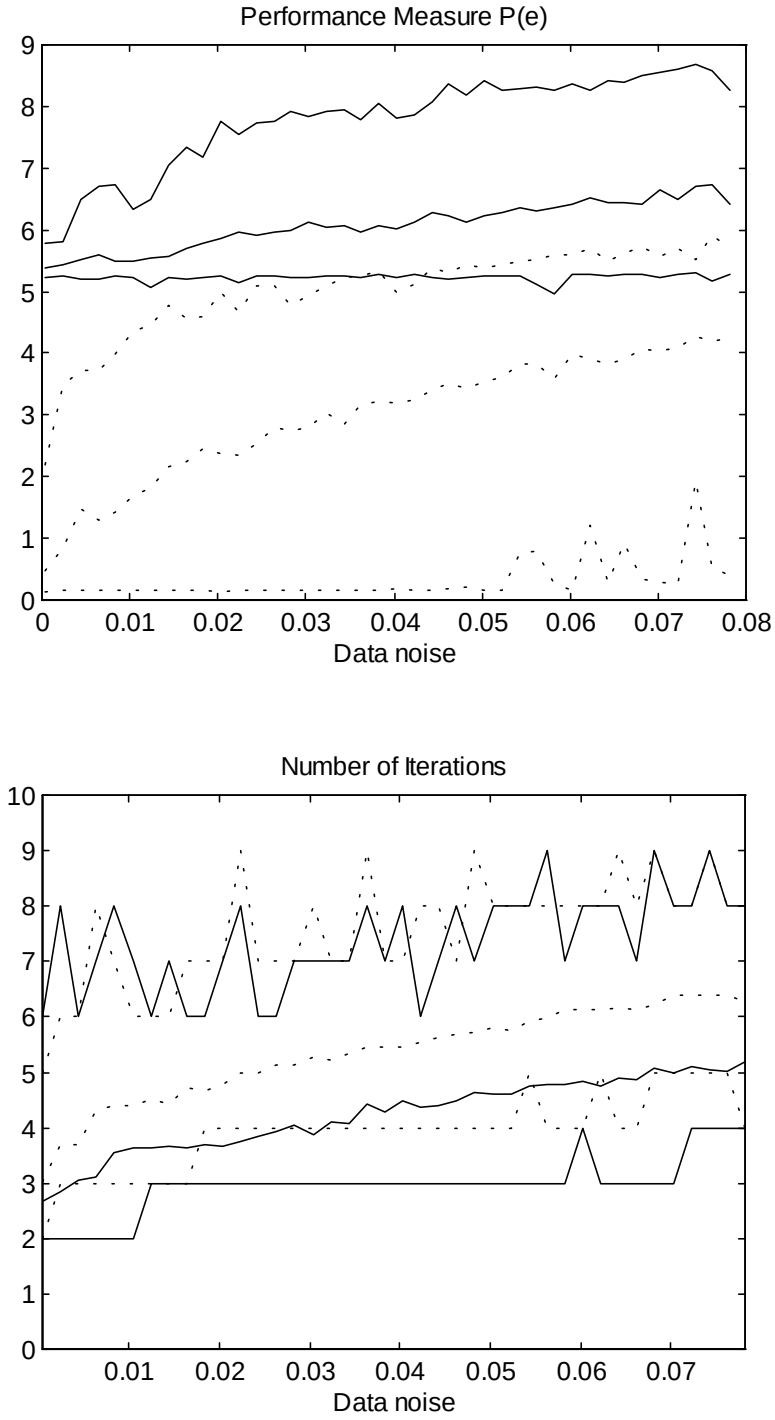

Figure 8: Values of the quality metrics $P(\mathbf{e})$ and N for the two test software implementations as a function of data noise. Results (minimum, mean and maximum) for Test Software A are shown using solid lines, and those for Test Software B using broken lines.

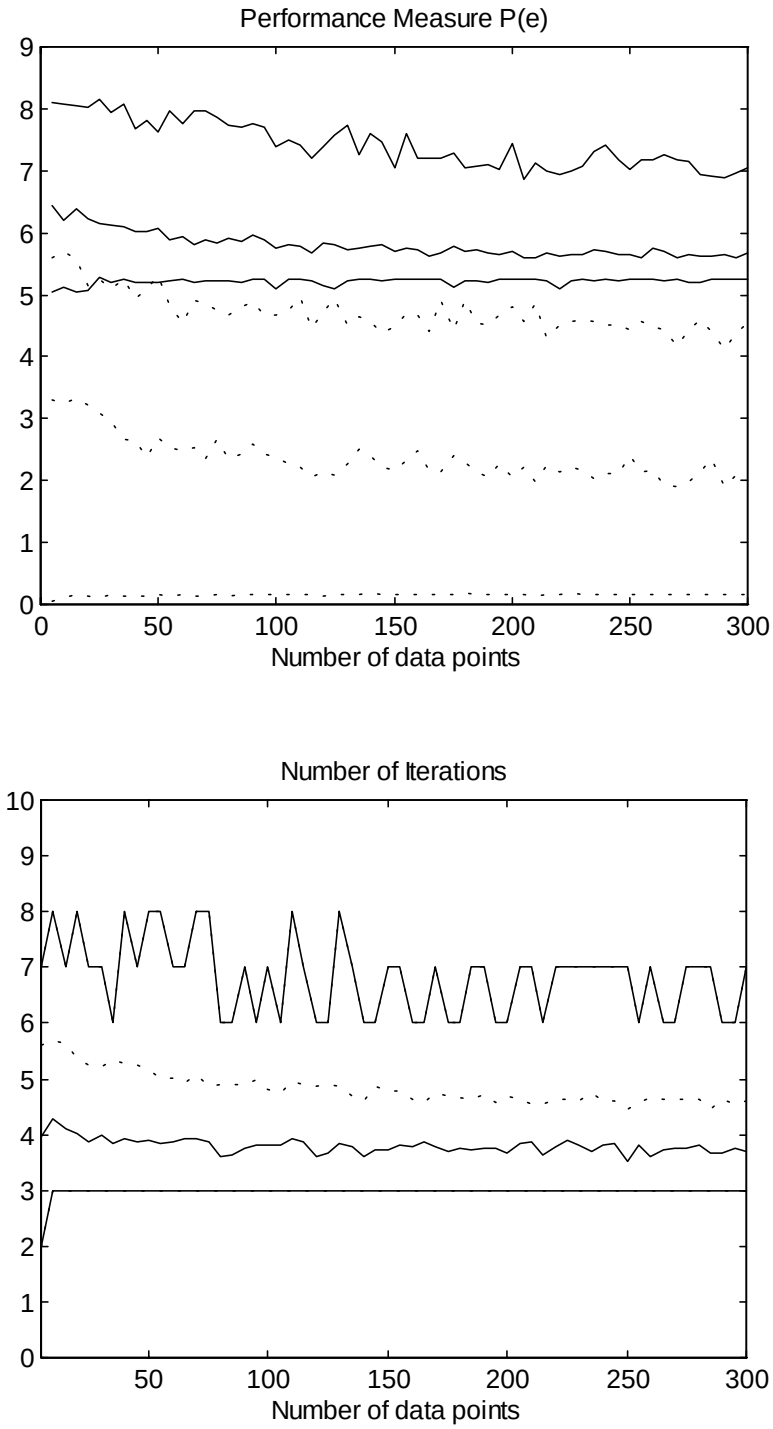

Figure 9: Values of the quality metrics $P(\mathbf{e})$ and N for the two test software implementations as a function of the number of data points. Results (minimum, mean and maximum) for Test Software A are shown using solid lines, and those for Test Software B using broken lines. (In the second of these figures, only the mean curve for Test Software B is visible: the minimum and maximum coincide with those for Test Software A.)

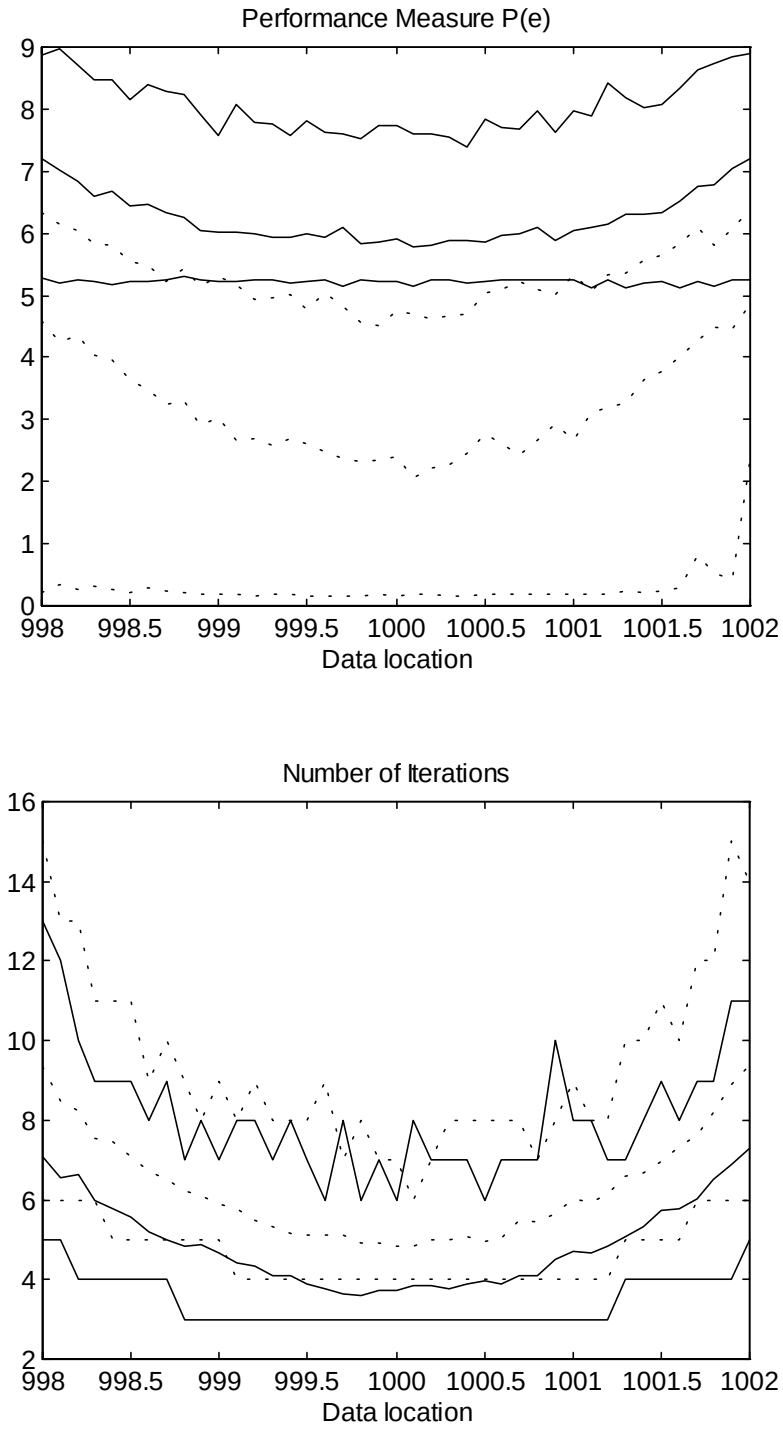

Figure 10: Values of the quality metrics $P(\mathbf{e})$ and N for the two test software implementations as a function of data location. Results (minimum, mean and maximum) for Test Software A are shown using solid lines, and those for Test Software B using broken lines.

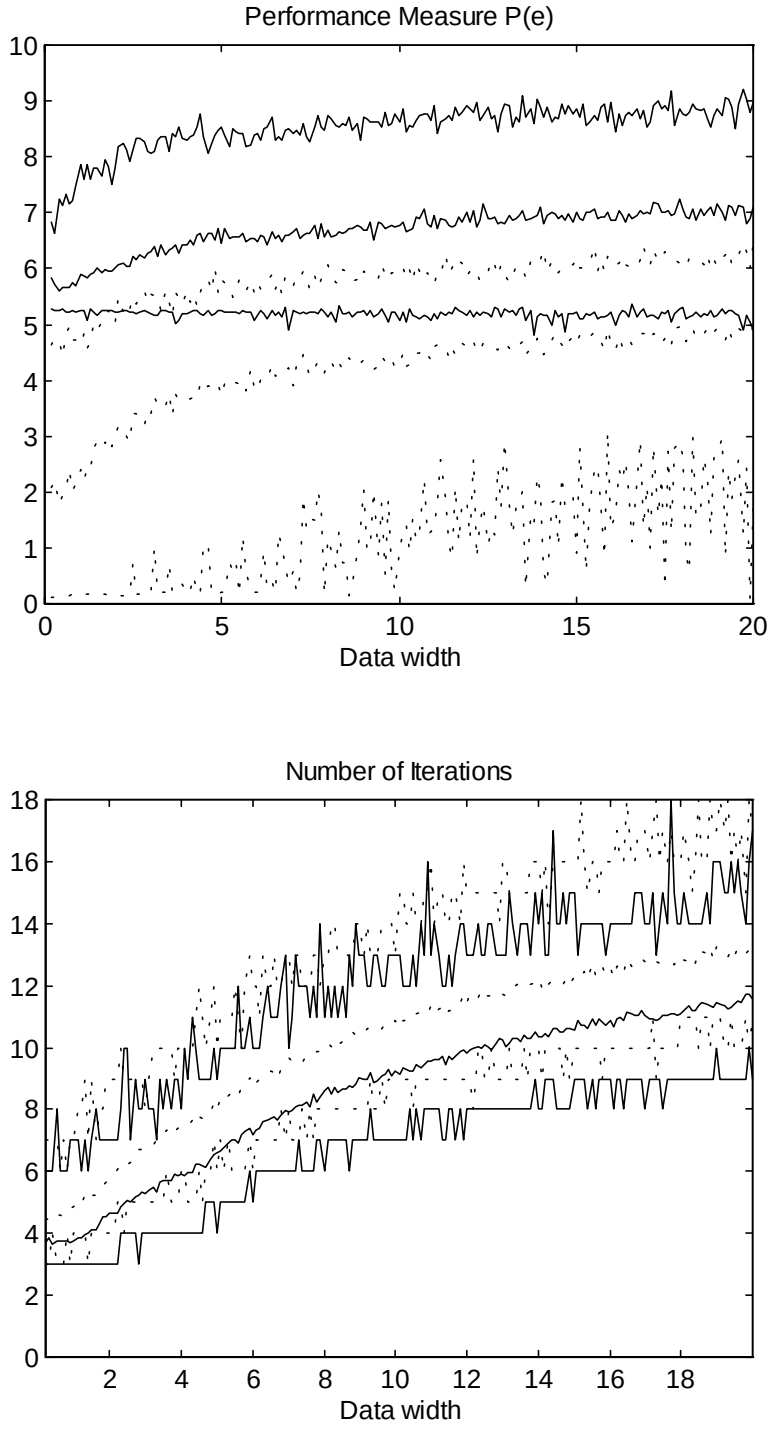

Figure 11: Values of the quality metrics $P(\mathbf{e})$ and N for the two test software implementations as a function of data width. Results (minimum, mean and maximum) for Test Software A are shown using solid lines, and those for Test Software B using broken lines.

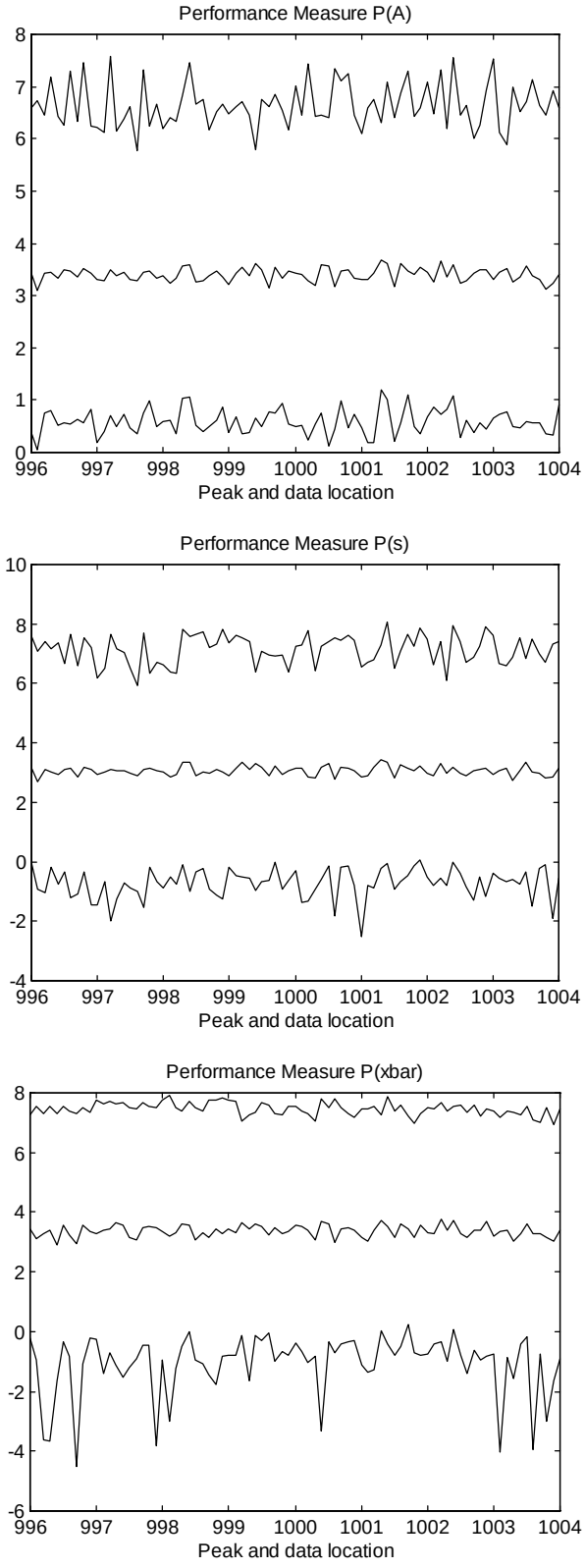

Figure 12: Values (minimum, mean and maximum) of the quality metrics $P(A)$, $P(s)$ and $P(\bar{x})$ for the two test software implementations as a function of peak and data location.

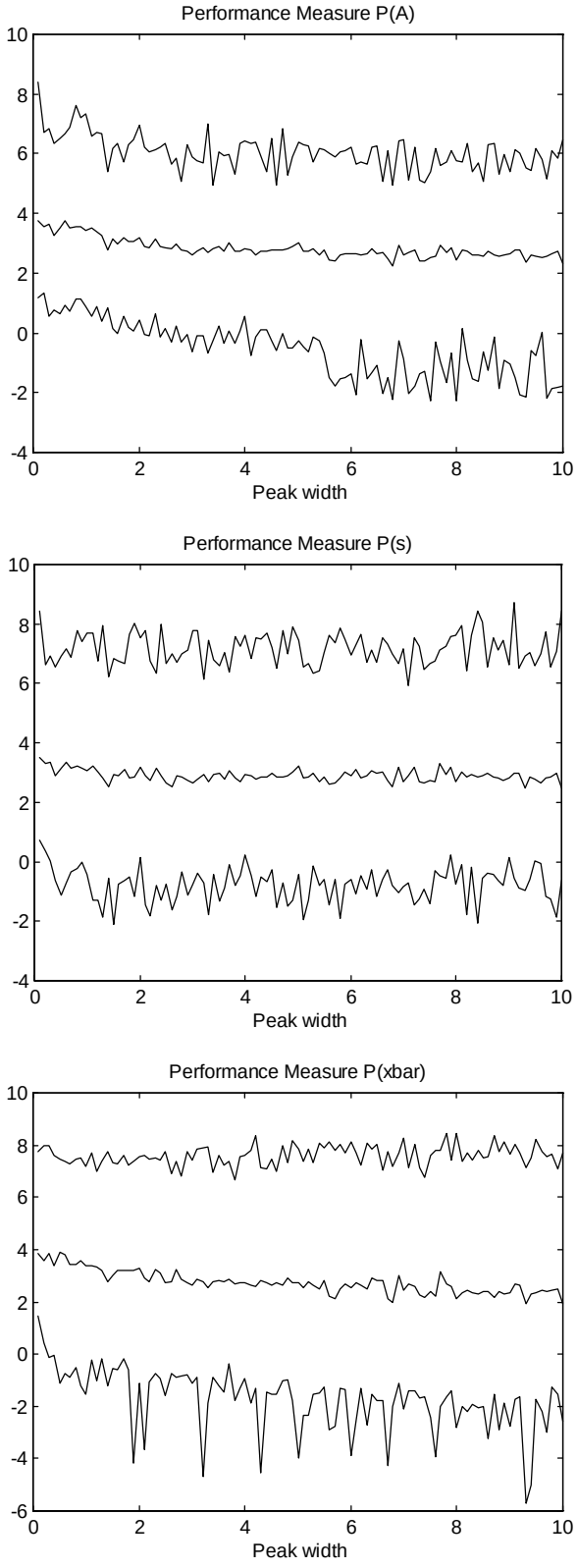

Figure 13: Values (minimum, mean and maximum) of the quality metrics $P(A)$, $P(s)$ and $P(\bar{x})$ for the two test software implementations as a function of peak width.

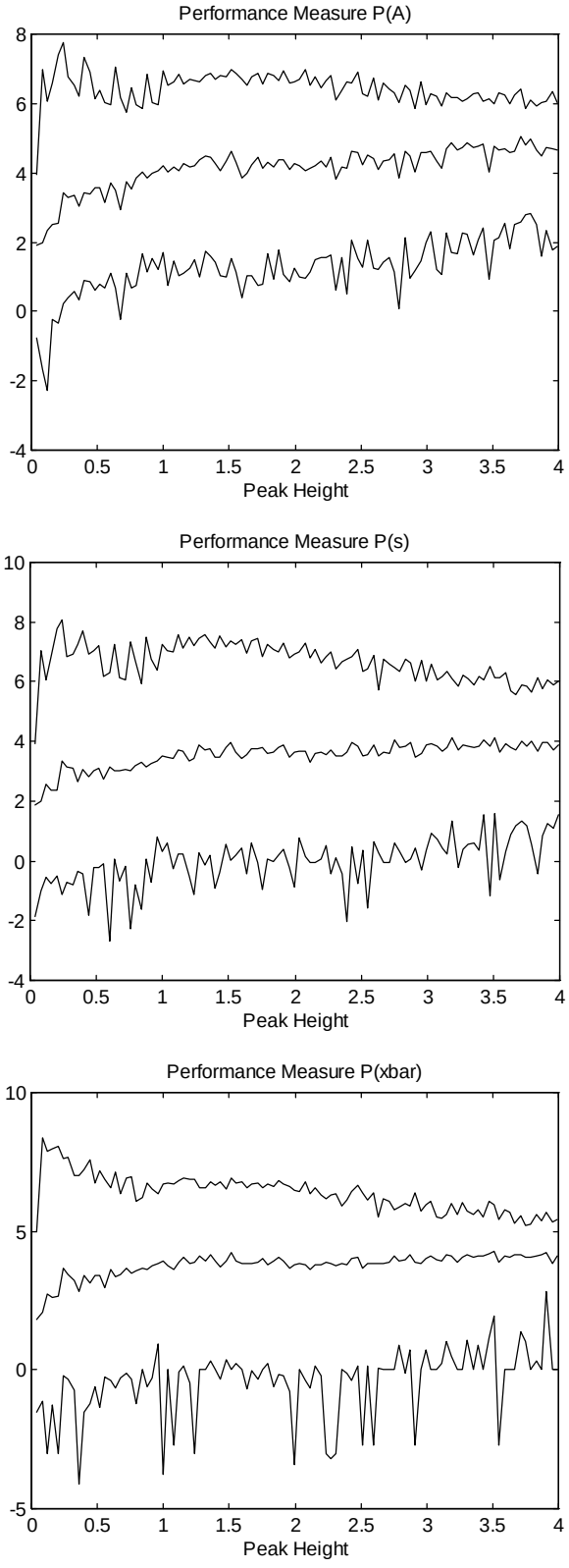

Figure 14: Values (minimum, mean and maximum) of the quality metrics $P(A)$, $P(s)$ and $P(\bar{x})$ for the two test software implementations as a function of peak height.

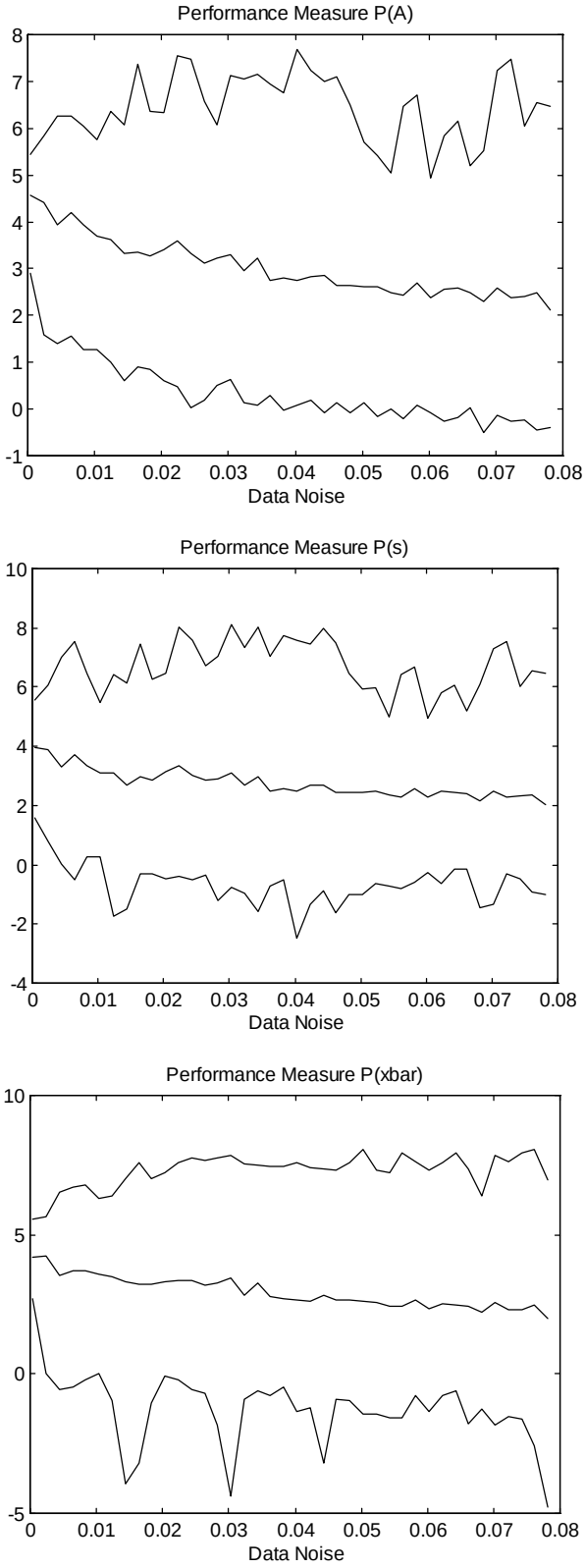

Figure 15: Values (minimum, mean and maximum) of the quality metrics $P(A)$, $P(s)$ and $P(\bar{x})$ for the two test software implementations as a function of data noise.

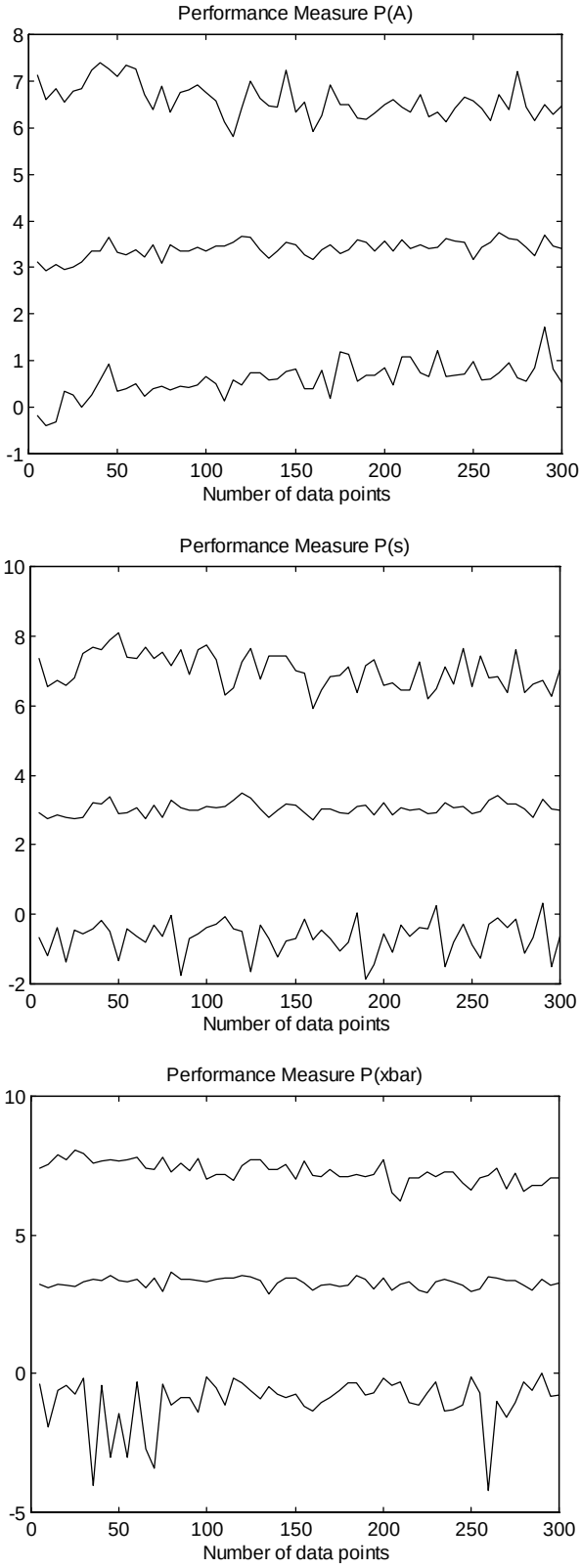

Figure 16: Values (minimum, mean and maximum) of the quality metrics $P(A)$, $P(s)$ and $P(\bar{x})$ for the two test software implementations as a function of the number of data points.

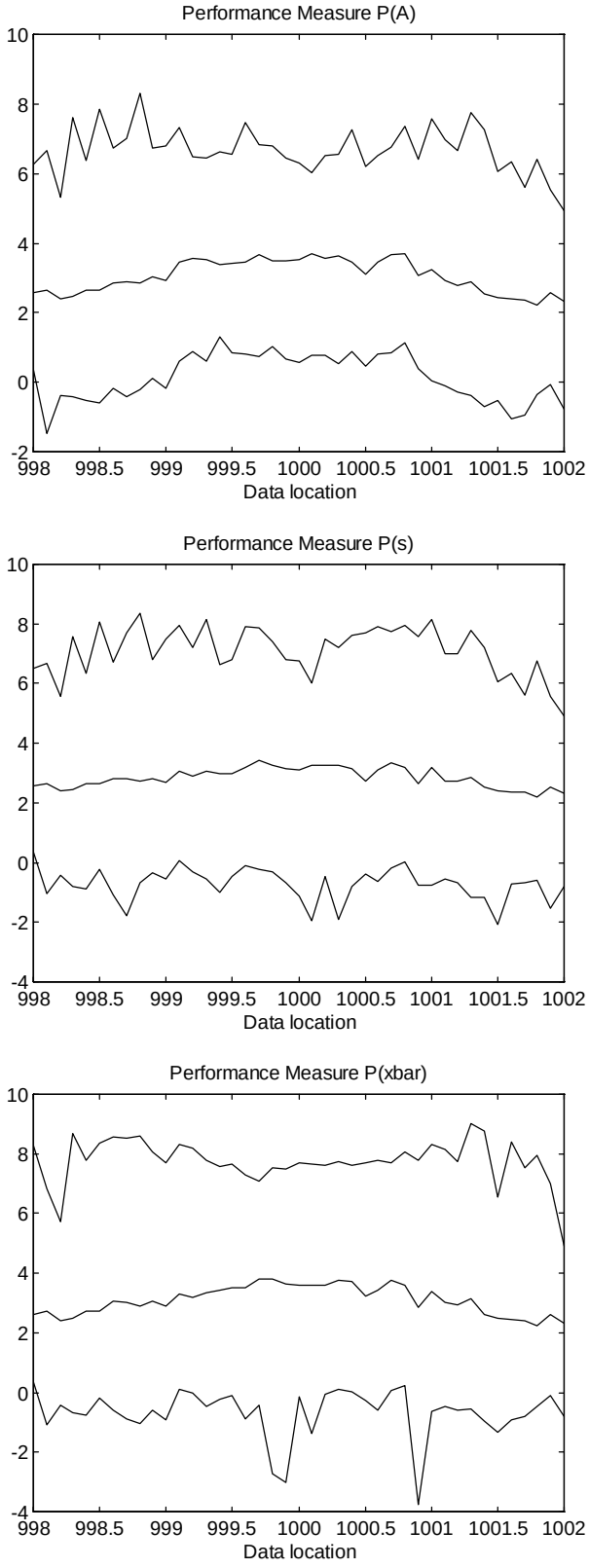

Figure 17: Values (minimum, mean and maximum) of the quality metrics $P(A)$, $P(s)$ and $P(\bar{x})$ for the two test software implementations as a function of data location.

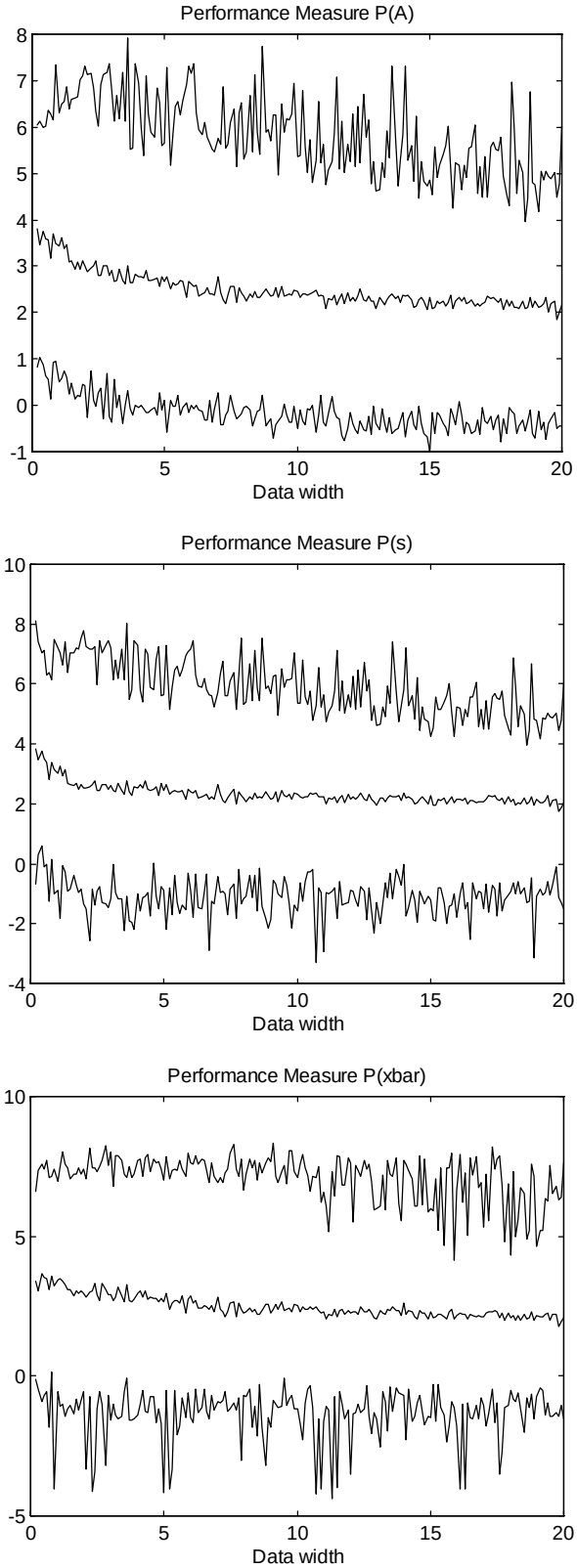

Figure 18: Values (minimum, mean and maximum) of the quality metrics $P(A)$, $P(s)$ and $P(\bar{x})$ for the two test software implementations as a function of data width.

6. References

[1] 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.

[2] 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.

[3] 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.

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

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