NPL Report CMSC 06/00

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

TESTING SPREADSHEETS AND OTHER PACKAGES USED IN METROLOGY

TESTING THE INTRINSIC FUNCTIONS OF S-PLUS

BY

J BARRETT and M P DAINTON

September 2000

Testing Spreadsheets and Other Packages Used in Metrology
Testing the Intrinsic Functions of S-PLUS

J Barrett and M P Dainton
Centre for Mathematics and Scientific Computing

September 2000

ABSTRACT

The aim of the work reported here is to contribute to the development of an infrastructure, comprising supporting information and guidelines, to ensure that the use of software, particularly spreadsheets and proprietary software packages, within metrology is made as effective as possible. This is to be achieved by reporting the results of the objective testing of the intrinsic and in-built functions included within spreadsheets and other proprietary software packages that are popular in metrology applications.

We describe the application of a general methodology for testing the numerical accuracy of scientific software to specific functions taken from the S-PLUS statistical software package. Each stage of the methodology, from the provision of a specification for each function tested through the definition of performance parameters and measures to the presentation and interpretation of the test results, is presented. In this way, and by stating any assumptions made in the application of the methodology, the testing undertaken is made as objective as possible.

This report constitutes one of the deliverables of Project 2.1 “Testing Spreadsheets and Other Packages Used in Metrology” within the UK Department of Industry’s National Measurement System *Software Support for Metrology* Programme 1998–2001.

ISSN 1471-0005

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

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

Contents

1. Introduction

The numerical correctness of scientific software, which is the issue addressed in this work, is important to metrology because a poor software implementation can lead to inaccuracy that could have been avoided, and as a consequence the accuracy of measurement results can be compromised. The aim of the work reported here is to contribute to the development of an infrastructure, comprising supporting information and guidelines, to ensure that the use of software, particularly spreadsheets and proprietary software packages, within metrology is made as effective as possible. This is to be achieved by reporting the results of the objective testing of the intrinsic and in-built functions included within spreadsheets and other proprietary software packages that are popular in metrology applications.

The work is primarily aimed at *users* of software packages within metrology applications. It is intended that the results presented will help users to understand whether for a particular application the functions used are fit for purpose, and to understand the limitations (if any) of those functions. In addition, however, it is anticipated that the work will be useful to the *developers* of the software, for example by highlighting any weaknesses identified in the functions tested.

The reports [1, 2] provide a review of

- the extent to which software packages are used within metrology, and
- the effort put into validating and testing these packages.

The report [1] is a general survey of the status of the mathematics and software used to support each metrology area in the UK National Measurement System (NMS), and [2] reports on existing work worldwide on spreadsheet quality and reliability with recommendations on its applicability in the metrology field.

Although there is some awareness of the potential limitations of spreadsheets and other software packages arising from the use of inaccurate or unstable numerical algorithms, relatively little testing and validation is performed on such software. Often, there is a tendency to *rely* upon well-established packages and to perform only a small number of checks using alternative methods of calculation. In contrast, bespoke software, possibly written using the same packages, is usually tested in accordance with software development procedures, for example, by comparing calculated results with reference results or results determined using other software.

In this report we present the results of testing undertaken on the in-built functions of the S-PLUS statistical software package. The report is a companion document to [10] and [11] that describe, respectively, similar testing of the Microsoft Excel and MathCAD software packages.

Some related work can be found in the literature. Notably, the papers [3, 4] report the results of assessing the reliability of three statistical packages, SAS, SPSS and S-PLUS, in the areas of estimation, random number generation and the calculation of statistical probabilities. Weaknesses are identified in all the random number generators, the S-PLUS correlation procedure, and in the one-way analysis of variance (ANOVA) and nonlinear least-squares routines of SAS and SPSS.

The report is organised as follows. In Section 2 we discuss the methodology underlying the testing carried out, and how it has been applied to the S-PLUS statistical software package. The methodology is described in detail in [8], and relies on the use of reference data sets and corresponding reference results to undertake black-box testing, as well as other approaches including self-consistency and continuity checks. Sections 3, 4 and 5 contain details of the implementation of the methodology for the testing of S-PLUS. Section 3 contains specifications of the S-PLUS functions tested. Section 4 describes the particular tests implemented and the

(so-called) performance measures that are used to quantify the performance of each function. In Section 5 we present the results of the testing, and provide some interpretation of these results. Section 6 contains our conclusions.

2. Methodology

A general methodology for evaluating the numerical accuracy of the results produced by scientific software is described in [5, 6, 7, 8] and illustrated by the case study [9] and its application to testing the Microsoft Excel [10] and MathCAD [11] software packages. 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 sets 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.

In addition to black-box testing of software using reference data sets and results, other complementary tests that do not require the availability of reference data are also useful and have been employed in this work. These tests include (see [8]):

a) consistency checks where, for example, we check the consistency of a forward calculation followed by an inverse calculation such as in comparing values of $\sin^{-1}\{\sin x\}$ against x;

b) continuity checks where, for example, for functions that use different evaluation methods over sub-ranges of the function arguments(s) we investigate the continuity of the evaluation methods across the sub-range boundaries;

c) spot checks against tabulated values;

d) checks of solution characterisations.

The methodology for testing that has been applied, and which is described in [8], comprises the following six stages:

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

The first two of these stages are usually carried out as part of the software development process, although in practice with varying amounts of formality. Stages 3 to 6 constitute the approach to software testing advocated in [8] with their application by a software *developer* presented in the case study [9]. The application of the methodology described here, however, is from the perspective of a *user* (of a proprietary software package). A user will usually not be part of the software development process and, consequently, stages 1 and 2 above are replaced by

1. *documenting* the specification of the test software, and
2. *interfacing* to the test software.

Recording the results of these stages is, nevertheless, important because it serves to define the basis of the testing undertaken and to make clear any assumptions made about the test software.

In addition to the test software itself, software is used for

a) generating reference data sets and corresponding reference results,

b) evaluating and presenting quality metrics and performance measures, and

c) applying complementary software tests (such as those listed above).

The generation of reference pairs, i.e., reference data sets and corresponding reference results, is described in Section 2.5. Reference pairs are generated using software implemented in Matlab [13] and employed in other testing work, such as in [9], or using the IMSL libraries supplied with the Digital Visual Fortran 90 software package [14]. The reference data and results are saved in data files that are subsequently imported into S-PLUS during the testing procedure (Section 2.2).

The evaluation of quality metrics and performance measures (Section 2.4), and their presentation as graphs (Section 2.6), is undertaken using S-PLUS functions. This approach enables a significant number of tests to be undertaken quickly.

In the remainder of this section we give an overview of the implementation of each of the six stages in the testing of S-PLUS intrinsic functions.

2.1 Documenting the specification of the test software

All S-PLUS intrinsic functions have an on-line help-file that describes the purpose of the function, together with details of inputs, outputs and any optional arguments, such as information on how to alter tolerance values for iterative methods. The help facility also provides examples of code necessary to run each function within the package. Dialog sheets can be found for the manipulation and processing of results from each function, and a listing of each function within these can be accessed on-line.

The information provided in the help-file is regarded in this work as providing the basis of a *specification* of the function against which the function is tested. As an example, consider the function LM[1] for fitting linear models. Quoting the help-file verbatim, it is stated that the function

> "…Returns an object that represents a fit of a linear model. The data to be fitted is stored in an S-PLUS data frame, and is called by the *formula* argument".

The inputs to the function are described thus[2]:

formula formula to be fitted, of the form:

$$y \sim f(x, a_1, \ldots, a_n)$$

where y denotes the column of the data frame in which the response data is stored, x denotes the column of the data frame in which the predictor data is stored, and $a_1, \ldots, a_n$ are the parameters to be estimated.

data data-frame containing the x and y data, and the data for the *subset* and *weights* arguments, if required. If this is missing, then the variables in the formula should be on the search list. This may also be a single number to handle some special cases - see below for details.

weights vector of observation weights; if supplied, the algorithm fits to minimize the sum of the weights multiplied into the squared residuals. The length of weights must be the same as the number of observations. The weights must be

[1] This stands for "Linear Model".

[2] Some knowledge of the S-PLUS system and its terminology is assumed here.

nonnegative and it is strongly recommended that they be strictly positive, since zero weights are ambiguous, compared to use of the subset argument.

subset expression saying which subset of the rows of the data should be used in the fit. This can be a logical vector (which is replicated to have length equal to the number of observations), or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.

na.action a function to filter missing data. This is applied to the *model.frame* after any subset argument has been used. The default (with *na.fail*) is to create an error if any missing values are found. A possible alternative is *na.omit*, which deletes observations that contain one or more missing values.

method the least squares fitting method to be used; the default is "qr". The method "*model.frame*" simply returns the model frame.

model logical flag: if TRUE, the model frame is returned in component model.

x logical flag: if TRUE, the model matrix is returned in component *x*.

y logical flag: if TRUE, the response is returned in component *y*.

qr logical flag: if TRUE, the QR decomposition of the model matrix is returned in component *qr*.

For the LM function (and for many of the other regression functions considered in this work), information about the algorithms implemented is provided. For the above example, it is clear that *least-squares* fitting to the data is undertaken using a QR decomposition of the model matrix. This information provides additional insight into the test software to help the user to make a judgement about the software based on the suitability of the algorithm implemented. For example, the use of a QR decomposition of the model matrix suggests that the LM function implements an algorithm that can be expected to have little induced numerical instability. The use of a black-box approach as described in [8] to testing software provides *quantitative* results to support such *qualitative* statements about the accuracy of the software.

The specification given above makes clear the requirements for generating reference data sets and corresponding results for testing the function. These are to generate reference data sets consisting of values for the inputs to the function together with corresponding reference results for the output computed from the inputs in accordance with the specification of the LM function described above.

The specification of each intrinsic function tested is constructed from its on-line help-file in a similar manner. Section 3 contains a summary of these specifications for all the functions tested.

2.2 Interfacing to the test software

The implementations of the functions tested in this work are the S-PLUS intrinsic functions contained within S-PLUS 4.0 [12]. Functions are accessed via an S-PLUS script file and applied to data that is, by default, stored in data frames. The script file is also used to control the calculation of quality metrics and the display of the results of the testing.

Where it is necessary to import data into S-PLUS from text files this is done in the following way:

1. Choose **File** menu.
2. Choose **Import data**.
3. Choose **From file**.

4. In the dialog box, choose the appropriate text file.
5. Click **Open**.

If the text file contains text before the first row of data, this is imported into S-PLUS as column headings. Otherwise text is assigned to a cell with the value "NA" in the data frame.

2.3 Specification of reference data sets

The aim of the software testing undertaken within the framework of the Software Support for Metrology programme is to verify whether the software is fit for purpose within the context of the National Measurement System. Reference data sets are required to reflect, as far as possible, the type of data sets that would commonly be encountered in practical measurement processes. It is an important consideration that when a function is tested, the range of inputs over which it is tested should reflect and include those of its intended use.

Performance parameters are used to capture the properties of data sets that would be encountered in practice and to describe the range of admissible inputs to the test software. By varying an individual performance parameter sequences of data sets may be generated, with the sequence forming a *graded* sequence in cases where the performance parameter relates directly to the condition or "degree of difficulty" of the problem represented by the data. By investigating the performance of the test software for such graded sequences, it is possible to identify cases where the test software is based upon a poor choice of mathematical algorithm.

For example, consider the problem addressed by the S-PLUS function LM for finding the least-squares best-fit gradient for a linear model

$$y = ax + b$$

to given data $\{(x_i, y_i): i = 1, \ldots, m\}$. Then, possible performance parameters include:

- the size m of data set,
- the reference value for the gradient a of the straight line model,
- the reference value for the intercept b of the straight line model, and
- the size of measurement error used to define the data values y_i, $i = 1, \ldots, m$.

Each of these parameters may be varied in turn, with the remaining parameters taking nominal values, and the performance of LM investigated as a function of each parameter. Where data sets are found for which the performance of the function is poor, these data sets can be described using the above performance parameters and related to properties of the user's metrology application. In this way, users are provided with information about the circumstances in which it is safe to use the function and when it is not.

Performance parameters are also used when applying the complementary software tests. For example, the consistency check based on the identity

$$\sin^{-1}\{\sin(x + 2n\pi)\} - x = 0,$$

is applied with m equispaced values of x in the range $[-\pi/2, +\pi/2]$, where m and n are regarded as performance parameters for the test.

Section 4 lists the performance parameters for each S-PLUS function and complementary test.

2.4 Specification of performance measures and testing requirements

Quality metrics are used to quantify the performance of the test software for the sequences and reference data sets to which the test software is applied. Furthermore, by relating the values of these metrics to the requirements of the user, it is possible to assess objectively whether the test software meets these requirements and is therefore "fit for purpose". Generally, the quality

metrics measure the departure of the test results returned by the test software from the reference results. The departure may be measured in (at least) three ways [8]:

1. the *absolute* difference between test and reference results, i.e.,

$$d_A(\mathbf{x}) = \left\| \mathbf{y}^{\text{test}} - \mathbf{y}^{\text{ref}} \right\|, \tag{1}$$

   where $\mathbf{x}$ denotes the input reference data set, and $\mathbf{y}^{\text{test}}$ and $\mathbf{y}^{\text{ref}}$ are, respectively, the test and reference results,

2. the *relative* difference between the test and reference results, i.e.,

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

3. a *performance measure* that accounts for factors such as the computational precision and conditioning of the problem. In [8] the following performance measure is derived:

$$P(\mathbf{x}) = \log_{10}\left(1 + \frac{1}{\kappa(\mathbf{x})\eta} \frac{\left\| \mathbf{y}^{\text{test}} - \mathbf{y}^{\text{ref}} \right\|}{\left\| \mathbf{y}^{\text{ref}} \right\|} \right), \tag{3}$$

   where $\kappa(\mathbf{x})$ measures the condition of the problem defined by the data set $\mathbf{x}$ [8], and η is the computational precision[3]. The performance measure $P(\mathbf{x})$ indicates the number of figures of accuracy lost by the test software over and above what software based on an optimally stable algorithm would produce.

For the complementary software tests, there are equivalent quality metrics to those described above. Suppose a consistency check is based on the identity

$$h(x) - x = 0,$$

where

$$h(x) = g(y) \qquad \text{and} \qquad y = f(x),$$

with g being the formal inverse of f. For example, for the consistency check based on the identity

$$\sin^{-1}\{\sin(x + 2n\pi)\} - x = 0,$$

we have

$$g(y) = \sin^{-1}\{y\} \qquad \text{and} \qquad f(x) = \sin(x + 2n\pi).$$

Then,

$$d_A(x) = |h(x) - x| \tag{4}$$

is an *absolute* measure of consistency between the functions f and g,

$$d_R(x) = \frac{|h(x) - x|}{|x|} \tag{5}$$

is a *relative* measure of consistency, and

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

$$P(x) = \log_{10}\left(1 + \frac{d_R(x)}{\eta}\right) \tag{6}$$

is a *performance measure* for consistency.

It is less easy to use the metrics (4), (5) and (6) to make *quantitative* statements about the test software, although *qualitative* judgements are possible. However, care is necessary: the result that the quality metric $d_A(x)$ given by (4) is zero does not imply that the individual functions *f* and *g* are working correctly, only that they form a consistent pair of functions for forward and inverse calculations. Consequently, it is important to supplement such checks with other tests, e.g., testing the individual functions *f* and *g* against other (possibly reference) implementations of the functions.

Section 4 lists the quality metrics for each S-PLUS function tested and each complementary test.

2.5 Generation of reference pairs

For the testing of regression functions, including the MEAN, VAR, LM and NLS functions (Section 3.1), for which a mathematical characterisation of the solution exists, *data generators* are used to construct reference data sets with known solutions, i.e., solutions specified *a priori*. The basis of the data generators are *null-space methods* [8] that use the solution characterisation to construct families or classes of data sets possessing nominally the same solution.

In the application of the complementary tests, particularly consistency checks, reference values are available from analytic considerations. For example, for the consistency check based on the identity

$$\sin^{-1}\{\sin(x + 2n\pi)\} - x = 0,$$

we can regard "x" as the reference data set, "$\sin^{-1}\{\sin(x + 2n\pi)\}$" as the test value returned by the software, and "x" as the reference value. This is consistent with the form of the quality metric for this test given in Section 2.4.

For many of the intrinsic functions tested, the test results returned are compared with values obtained from other implementations. The software employed for this purpose is contained within the IMSL libraries supplied with the Digital Visual Fortran 90 software package [14]. These particular libraries were chosen for convenience and because the functions included within them are based on reliable and established numerical algorithms developed over many years. Of course there are other sources of high-quality software that could be used for this purpose: for example, the NAG library. It is proposed that as part of future work the sources of software used for testing will be expanded to include other such libraries.

2.6 Presentation and interpretation of performance measures

Having applied the test software to a reference data set to obtain a test result, the test result is compared with the reference result corresponding to the data set by computing a quality metric or performance measure such as is defined by (1), (2) or (3) (Section 2.4). The quality metrics are presented as functions of each (one or more) performance parameter (Section 2.3) either in tabular form or as a graph against the performance parameter, i.e., as a performance profile. Plotting of results in S-PLUS can be done by use of the *plot* function within a script program. This permits the presentation of results to be fully automated and integrated into the testing procedure.

To use the testing results, for example, where presented in the form of a performance profile, the user needs to

1. decide the range of values of the performance parameter that correspond to the application, and hence identify that part of the performance profile appropriate to the application, and
2. decide whether the values of the quality metric over the identified range of the performance profile meet the accuracy requirements of the application.

In addition, by examining the performance over the complete range of the performance parameter, statements can be made about the general performance of the test software with respect to this parameter.

For example, one of the performance parameters used in the testing of the LM function is the signal-to-noise ratio. A graph is provided showing values of the performance measure $P(\mathbf{x})$ given by (3) against this parameter. The graph can be used

- to identify values of the signal-to-noise ratio for which the performance of the LM function meets a specified accuracy requirement, e.g., the results are to be accurate to a specified number of significant figures, or
- to assess the performance of the LM function for data sets characterised by a particular range of signal-to-noise ratio values.

For the complementary software tests, quality metrics, such as those defined by (4), (5) and (6) (Section 2.4), are presented (in tabular or graphical form) as a function of the input argument for various choices of the performance parameters (Section 2.3). For example, for the consistency check based on the identity

$$\sin^{-1}\{\sin(x + 2n\pi)\} - x = 0,$$

with $x \in [-\pi/2, +\pi/2]$, we show a plot of the performance measure $P(x)$ against x for various choices of the integer n. In addition to interpreting the value of the quality metric over sub-ranges of x of interest, considerations include the extent to which the quality metric varies smoothly with x, and the identification of the presence and position of extreme points and discontinuities in the metric.

3. Specifications of the Intrinsic Functions

In this section we list the S-PLUS worksheet functions that have been tested as part of this work, and provide specifications for these functions. The description of each function tested is a verbatim quotation of the on-line help documentation provided with S-PLUS. Where appropriate, our interpretations of these descriptions are included as footnotes. The functions are grouped into the following classes:

1. regression functions,
2. mathematical and trigonometric functions, and
3. statistical distributions.

This grouping reflects the intended purpose of the functions as well as the methods used to test them (Section 4). We include the MEAN and VAR worksheet functions within the class of regression functions because their outputs can be related to the problem of finding the least-squares best-fit constant to the set of sample values and, consequently, the generation of reference data sets and corresponding reference results can be performed in a similar way to the other regression functions listed.

Specifications for the functions tested from each of the above classes are listed in, respectively, Sections 3.1, 3.2 and 3.3. The particular implementations of the functions tested in this work are those contained within S-PLUS 4.0 [12].

3.1 Regression Functions

MEAN

Returns a number which is the mean[4] of the data. A fraction to be trimmed from each end of the ordered data can be specified.

mean(*x*, *trim*=0, *na.rm*=F)

REQUIRED ARGUMENTS

x numeric object. Missing values (NAs) are allowed.

OPTIONAL ARGUMENTS

trim fraction (between 0 and .5, inclusive) of values to be trimmed from each end of the ordered data.[5] If *trim* = .5, the result is the median.

na.rm logical flag: should missing values be removed before computation?

VAR and **COR**

Returns the variance of a vector, the variance-covariance (or correlation) matrix of a data matrix, or covariances between matrices or vectors. A trimming fraction may be specified for correlations. *cor*() returns correlations, and *var*() returns variances and covariances or sums of squares. If *x* is a matrix, the result is a matrix such that the [i,j] element is the covariance (correlation) of *x*[,i] and either *y*[,j] or *x*[,j]. If *x* is a vector, the result is a vector, with length equal to the number of columns of *y* (or length 1 if *y* is not supplied).

var(*x*, *y*, *na.method*="fail", *unbiased*=T, *SumSquares*=F)

cor(*x*, *y*, *trim*=0, *na.method*="fail", *unbiased*=T)

REQUIRED ARGUMENTS

x numeric matrix or vector, or data frame. May be complex. If a matrix, columns represent variables and rows represent observations. If a data frame, non-numeric variables result in missing values in the result.

OPTIONAL ARGUMENTS

y numeric matrix or vector, or data frame. May be complex. If a matrix, columns represent variables and rows represent observations. If a data frame, non-numeric variables result in missing values in the result. This must have the same number of observations as *x*.

trim a number less than .5 giving the proportion trimmed in the internal calculations for *cor*. This should be a number larger than the suspected fraction of outliers.

na.method a character string specifying how missing values are to be handled. Options are

"fail" (stop if any missing data is found),

"omit" (omit rows with any missing data),

"include" (missing values in the input result in missing values in the output)

"available" (use available observations, see below). Only enough of the string to determine a unique match is required.

---

[4] Assumed to be the arithmetic mean.

[5] The trimming operation defined here is ambiguous. For the case *trim* < 0.5, an interpretation is that *n* largest and *n* smallest values in the sample are ignored, where *n* is the integer part of *trim* × length(*x*). For the case *trim* = 0.5, the median is returned as stated.

unbiased if TRUE, then variances are sample variances, e.g.

sum((x-mean(x))^2)/(N-1)

for a vector of length *N*, which is unbiased if the values in *x* are obtained by simple random sampling[6]. If FALSE, the definition

sum((x-mean(x))^2)/N

is used instead.

SumSquares if TRUE, then unnormalized sums of squares are returned, with no division by either N or $(N - 1)$ (and unbiased is ignored).

LM

Returns an object that represents a fit of a linear model. The data to be fitted is stored in an S-PLUS data frame, and is called by the *formula* argument.

lm(*formula, data, weights, subset, na.action, method, model, x, y, contrasts, ...*)

REQUIRED ARGUMENTS

formula formula to be fitted, of the form:

$y \sim f(x, a_1, \ldots, a_n)$

where *y* denotes the column of the data frame in which the response data is stored, *x* denotes the column of the data frame in which the predictor data is stored, and $a_1, \ldots, a_n$ are the parameters to be estimated.

OPTIONAL ARGUMENTS

data data frame containing the *x* and *y* data, and the data for the *subset* and *weights* arguments, if required. If this is missing, then the variables in the formula should be on the search list. This may also be a single number to handle some special cases - see below for details.

weights vector of observation weights; if supplied, the algorithm fits to minimize the sum of the weights multiplied into the squared residuals. The length of *weights* must be the same as the number of observations. The weights must be nonnegative and it is strongly recommended that they be strictly positive, since zero weights are ambiguous, compared to use of the *subset* argument.

subset expression saying which subset of the rows of the data should be used in the fit. This can be a logical vector (which is replicated to have length equal to the number of observations), or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.

na.action a function to filter missing data. This is applied to the *model.frame* after any subset argument has been used. The default (with *na.fail*) is to create an error if any missing values are found. A possible alternative is *na.omit*, which deletes[7] observations that contain one or more missing values.

method the least squares fitting method to be used; the default is "qr". The method "*model.frame*" simply returns the model frame.

model logical flag: if TRUE, the model frame is returned in component *model*.

[6] In other words, an unbiased estimate of the population variance is returned.

[7] In other words, does not use.

x logical flag: if TRUE, the model matrix is returned in component *x*.

y logical flag: if TRUE, the response is returned in component *y*.

qr logical flag: if TRUE, the QR decomposition of the model matrix is returned in component *qr*.

contrasts a list giving contrasts for some or all of the factors appearing in the model formula. The elements of the list should have the same name as the variable and should be either a contrast matrix (specifically, any full-rank matrix with as many rows as there are levels in the factor), or else a function to compute such a matrix given the number of levels.

... additional arguments for the fitting routines (see *lm.fit* and the functions it calls). Two possibilities are *singular.ok*=T to instruct the fitting to continue in the presence of over-determined models, and tolerance (default 1e–07) to change the tolerance for determining when models are over-determined[8].

NLS

Fits a nonlinear regression model via least squares. It returns an object containing the parameters, residuals, fitted values, and derivatives of the model at the end of the iteration[9].

nls(*formula, data, start, control, algorithm, trace*)

REQUIRED ARGUMENTS

formula formula to be fitted, of the form:

$$y \sim f(x, a_1, \ldots, a_n)$$

where *y* denotes the column of the data frame in which the response data is stored, *x* denotes the column of the data frame in which the predictor data is stored, and $a_1, \ldots, a_n$ are the parameters to be estimated.

data a data frame in which to do the computations. In addition to the usual data variables, the data frame may contain parameters (set, typically, by using the assignment form of parameters or *param*) that establish initial values for the model parameters.

OPTIONAL ARGUMENTS

start a list or numerical vector. Although it is optional, use of *start* is recommended for unambiguous specification of the parameters. If *start* is omitted, the assumption is that any names occurring in formula that are not variables in the data frame are parameters. The list form of *start* allows the individual parameter names to refer to subsets of the parameters of arbitrary length. If a numeric starting vector is supplied, the named parameters must each be of length 1. In the case of partially linear models (*algorithm*="plinear"), only the nonlinear parameters should be supplied.

control list of control values to be used in the iteration. See *nls.control* for the possible control options, their default settings, and their effect on the fitting.

algorithm character string stating which algorithm to use. The default algorithm is a Gauss-Newton algorithm. If algorithm is "plinear" the Golub-Pereyra algorithm for partially linear least-squares models is used.

[8] It is not stated to which quantity this tolerance is applied.

[9] Or rather, at the end of the entire iterative process.

trace logical flag or character string naming a tracing function. If *trace* is FALSE, then no tracing is performed. If *trace* is TRUE, then *trace.nls* is used to trace. See *trace.nls* for the relevant arguments if you want to write your own tracer. Note, however, that the special accumulation technique in *trace.nls* depends on co-operation with *nls* that is switched on only by setting the *trace* argument to TRUE.

3.2 Mathematical and Trigonometric Functions

ACOS

acos(*z*) returns the inverse cosine of *z* in radians, where *z* can be real or complex. For real *z*, the domain is the interval [−1,1], and the range is $0 \le acos(z) \le \pi$.

ACOSH

acosh(*z*) returns the inverse hyperbolic cosine of *z*, where *z* can be real or complex.

ASIN

asin(*z*) returns the inverse sine of *z* in radians, where *z* can be real or complex. For real *z*, the domain is the interval [−1,1], and the range is $-\pi/2 \le asin(z) \le \pi/2$.

ASINH

asinh(*z*) returns the inverse hyperbolic sine of *z*, where *z* can be real or complex.

ATAN

atan(*z*) returns the inverse tangent of *z* in radians, where *z* can be real or complex. For real *z*, the domain is unrestricted, and the range is $-\pi/2 < atan(z) < \pi/2$.

atan(*y*,*x*) returns the angle in radians from the positive *x*-axis to the point (*x*, *y*) in the *xy* plane, where *x* and *y* are real numbers. The domain is unrestricted, and the range is $-\pi < atan(y, x) \le \pi$.

ATANH

atanh(*z*) returns the inverse hyperbolic tangent of *z*, where *z* can be real or complex. Real arguments must be less than 1 in absolute value.

COS

cos(*z*) returns the cosine of *z*, where *z* can be real (in radians) or complex.

COSH

cosh(*z*) returns the hyperbolic cosine of *z*, where *z* can be real (in radians) or complex.

EXP

exp(*z*) returns the exponential of *z*, where *z* can be real or complex.

LOG

log(*z*) returns the natural logarithm of *z* to base *e*, where *z* can be real or complex. Real arguments must be nonnegative.

log(*z, base*) returns the logarithm of *z* to base *b*, where *z* can be nonnegative real or complex, and *base* can be positive real or complex.

LOG10

log10(*z*) returns the common logarithm of *z* to base 10, where *z* can be real or complex. Real arguments must be nonnegative.

SIN

sin(*z*) returns the sine of *z*, where *z* can be real (in radians) or complex.

SINH

sinh(*z*) returns the hyperbolic sine of *z*, where *z* can be real (in radians) or complex.

TAN

tan(*z*) returns the tangent of *z*, where *z* can be real (in radians) or complex.

TANH

tanh(*z*) returns the hyperbolic tangent of *z*, where *z* can be real (in radians) or complex.

3.3 Statistical Distributions

Beta distribution

Functions for the density, cumulative distribution, and quantiles of the Beta distribution.

dbeta(*x*, *shape1*, *shape2*)

pbeta(*q*, *shape1*, *shape2*, *ncp*=0)

qbeta(*p*, *shape1*, *shape2*)

REQUIRED ARGUMENTS

x	vector of quantiles. Missing values (NAs) are allowed.
q	vector of quantiles. Missing values (NAs) are allowed.
p	vector of probabilities. Missing values (NAs) are allowed.
shape1	vector of (positive) shape parameters[10]. This is replicated to be the same length as *p* or *q* or the number of deviates generated[11].
shape2	vector of (positive) shape parameters. This is replicated to be the same length as *p* or *q* or the number of deviates generated.

OPTIONAL ARGUMENTS

ncp	vector of non-negative noncentrality parameters.

Binomial distribution

Functions for the density, cumulative probability, and quantiles for the binomial discrete distribution. The quantile is defined as the smallest value *q* such that Prob(Binomial random variate <= *x*) ≥ *p*.

dbinom(*x*, *size*, *prob*)

pbinom(*q*, *size*, *prob*)

qbinom(*p*, *size*, *prob*)

OPTIONAL ARGUMENTS

x	vector of quantiles. Missing values (NAs) are allowed.

[10] The Beta distribution takes real values between 0 and 1. Special cases of the Beta distribution are the Uniform[0,1] distribution (see Uniform) and the arcsin distribution (for which *shape1* = *shape2* = 0.5).

[11] In the case that *shape1* and *shape2* are provided as single scalars. This interpretation applies also to optional arguments used in the same way for other functions.

q vector of (positive) quantiles (number of successes obtained in *size* binomial trials with probability *prob* of success). Missing values (NAs) are allowed.

p vector of probabilities. Missing values (NAs) are allowed.

size vector of (positive integer) numbers of coin flips for which the Binomial distribution measures the number of heads.

prob vector of probabilities of a head.

Chi-square distribution

Functions for the density, cumulative probability, and quantiles and for the chi-square distribution.

dchisq(*x*, *df*)

pchisq(*q*, *df*, *ncp*=0)

qchisq(*p*, *df*)

REQUIRED ARGUMENTS

x vector of quantiles. Missing values (NAs) are allowed.

q vector of (positive) quantiles. Missing values (NAs) are allowed.

p vector of probabilities. Missing values (NAs) are allowed.

df degrees of freedom (> 0). This is replicated to be the same length as *p* or *q* or the number of deviates generated. Non-integer values are allowed, but missing values are not.

OPTIONAL ARGUMENTS

ncp vector of non-negative numbers giving the noncentrality parameter.

Exponential distribution

Functions for the density, cumulative probability, and quantiles for the exponential distribution.

dexp(*x*, *rate*=1)

pexp(*q*, *rate*=1)

qexp(*p*, *rate*=1)

REQUIRED ARGUMENTS

x vector of quantiles. Missing values (NAs) are allowed.

q vector of quantiles. Missing values (NAs) are allowed.

p vector of probabilities. Missing values (NAs) are allowed.

OPTIONAL ARGUMENTS

rate the inverse of the mean of the distribution.

F distribution

Functions for the density, cumulative probability, and quantiles for the F distribution.

df(*x*, *df1*, *df2*)

pf(*q*, *df1*, *df2*, *ncp*=0)

qf(*p*, *df1*, *df2*)

REQUIRED ARGUMENTS

x vector of (positive) quantiles. Missing values (NAs) are allowed.

q vector of (positive) quantiles. Missing values (NAs) are allowed.

p vector of probabilities. Missing values (NAs) are allowed.

df1 degrees of freedom for the numerator. This is replicated to be the same length as *p* or *q* or the number of deviates generated. Non-integer values are allowed, but missing values are not.

df2 degrees of freedom for the denominator. This is replicated to be the same length as *p* or *q* or the number of deviates generated. Non-integer values are allowed, but missing values are not.

OPTIONAL ARGUMENTS

ncp vector of positive numbers giving the noncentrality parameter.

Gamma distribution

Functions for the density, cumulative probability, and quantiles for the gamma distribution.

dgamma(*x*, *shape*, *rate*=1)

pgamma(*q*, *shape*, *rate*=1)

qgamma(*p*, *shape*, *rate*=1)

REQUIRED ARGUMENTS

x vector of (positive) quantiles. Missing values (NAs) are allowed.

q vector of (positive) quantiles. Missing values (NAs) are allowed.

p vector of probabilities. Missing values (NAs) are allowed.

shape shape parameter (> 0). This is replicated to be the same length as *p* or *q* or the number of deviates generated.

OPTIONAL ARGUMENTS

rate inversely proportional to the mean of the distribution; often called *lambda*. The mean of the distribution is *shape*/*rate*, the variance is *shape*/*rate*^{2}, and the skewness is 2/√*shape*.

Gamma function

Returns the gamma function or the natural logarithm of the gamma function.

gamma(*x*)

lgamma(*x*)

REQUIRED ARGUMENTS

x numeric or complex object. Missing values (NAs) are allowed.

Hypergeometric distribution

Functions for the density, cumulative probability, and quantiles for the Hypergeometric discrete distribution.

dhyper(*q*, *m*, *n*, *k*)

phyper(*q*, *m*, *n*, *k*)

qhyper(*p*, *m*, *n*, *k*)

REQUIRED ARGUMENTS

q vector of values of a random variable representing the number of red balls out of a sample of size *k* drawn from an urn containing *m* red balls and *n* black ones.

p vector of probabilities. Missing values (NAs) are allowed. Its values must be between 0 and 1.

m number of red balls in the urn. This could be a vector with non-negative integer elements.

n number of black balls in the urn. This could also be a vector with non-negative integer elements.

k number of balls drawn from an urn with *m* red and *n* black balls. This can be a vector like *m* and *n*.

Lognormal distribution

Functions for the density, cumulative probability, and quantiles for the lognormal distribution.

dlnorm(*x*, *meanlog*=0, *sdlog*=1)

plnorm(*q*, *meanlog*=0, *sdlog*=1)

qlnorm(*p*, *meanlog*=0, *sdlog*=1)

REQUIRED ARGUMENTS

x vector of (positive) quantiles. Missing values (NAs) are allowed.

q vector of (positive) quantiles. Missing values (NAs) are allowed.

p vector of probabilities. Missing values (NAs) are allowed.

OPTIONAL ARGUMENTS

meanlog, *sdlog* vectors of means and standard deviations of the distribution of the log of the random variable. Thus, *exp*(*meanlog*) is a scale parameter and *sdlog* is a shape parameter for the lognormal distribution. These are replicated to be the same length as *p* or *q* or the number of deviates generated. Missing values are not accepted except in *dlnorm*.

Normal distribution

Functions for the density, cumulative probability, and quantiles for the normal (also called Gaussian) distribution.

dnorm(*x*, *mean*=0, *sd*=1)

pnorm(*q*, *mean*=0, *sd*=1)

qnorm(*p*, *mean*=0, *sd*=1)

REQUIRED ARGUMENTS

x vector of quantiles. Missing values (NAs) are allowed.

q vector of quantiles. Missing values (NAs) are allowed.

p vector of probabilities. Missing values (NAs) are allowed.

OPTIONAL ARGUMENTS

mean vector of means. This is replicated to be the same length as *p* or *q* or the number of deviates generated.

sd vector of (positive) standard deviations. This is replicated to be the same length as *p* or *q* or the number of deviates generated.

Poisson distribution

Functions for the density, cumulative distribution, and quantiles of the Poisson distribution. The quantile is defined as the smallest value *q* such that Prob(Poisson random variate <= *x*) ≥ *p*.

dpois(*x*, *lambda*)

ppois(*q*, *lambda*)

qpois(*p*, *lambda*)

REQUIRED ARGUMENTS

x vector of (positive) quantiles. Missing values (NAs) are allowed.

q vector of (positive) quantiles. Missing values (NAs) are allowed.

p vector of probabilities. Missing values (NAs) are allowed.

lambda vector of (positive) means.

Student's t distribution

Functions for the density, cumulative probability, and quantiles for the Student's t distribution.

dt(*x*, *df*)

pt(*q*, *df*)

qt(*p*, *df*)

REQUIRED ARGUMENTS

x vector of quantiles. Missing values (NAs) are allowed.

q vector of quantiles. Missing values (NAs) are allowed.

p vector of probabilities. Missing values (NAs) are allowed.

df vector of degrees of freedom. This is replicated to be the same length as *p* or *q* or the number of deviates generated.

Weibull distribution

Functions for the density, cumulative distribution, and quantiles of the Weibull distribution.

dweibull(*x*, *shape*, *scale*=1)

pweibull(*q*, *shape*, *scale*=1)

qweibull(*p*, *shape*, *scale*=1)

REQUIRED ARGUMENTS

x vector of (positive) quantiles. Missing values (NAs) are allowed.

q vector of (positive) quantiles. Missing values (NAs) are allowed.

p vector of probabilities. Missing values (NAs) are allowed.

shape vector of (positive-valued) shape parameters.

OPTIONAL ARGUMENTS

scale vector of (positive-valued) scale parameters.

4. Specification of Performance Parameters and Measures

4.1 Regression Functions

For the testing of the regression functions within S-PLUS we followed the methodology of generating reference data sets and results using data generators described in [8]. Reference data sets with corresponding reference results are readily constructed, and a quantitative quality metric is explicitly available [8, 9]. We present here for each regression function tested the performance parameters used to define the reference data sets and the performance measures used to compare the results returned by the function tested with the reference results.

MEAN

Performance parameters for this function include

- the reference sample mean,
- the number of points in the sample, and
- the reference sample standard deviation.

The performance measure $P(\bar{x})$ used to measure the departure of the computed sample mean $\bar{x}$ returned by the test software from the reference sample mean $\bar{x}^{\text{ref}}$ is given by

$$P(\bar{x}) = \log_{10}\left(1 + \frac{1}{\kappa(\bar{x})\eta}\frac{\left|\bar{x} - \bar{x}^{\text{ref}}\right|}{\left|\bar{x}^{\text{ref}}\right|}\right), \tag{7}$$

where $\kappa(\bar{x})$ measures the relative conditioning of the problem,

$$\kappa(\bar{x}) = \frac{s}{\left|\bar{x}^{\text{ref}}\right|},$$

and η is the machine precision [8].

VAR

Performance parameters for this function include

- the reference sample mean,
- the number of points in the sample, and
- the reference sample standard deviation.

The function VAR returns the variance of the sample, and its square root is the sample standard deviation. The performance measure $P(s)$ used to measure the departure of the computed sample standard deviation s returned by the test software from the reference sample standard deviation s^{ref} is given by

$$P(s) = \log_{10}\left(1 + \frac{1}{\kappa(s)\eta}\frac{\left|s - s^{\text{ref}}\right|}{\left|s^{\text{ref}}\right|}\right), \tag{8}$$

where $\kappa(s)$ measures the relative conditioning of the problem,

$$\kappa(s) = \frac{\left|\bar{x}\right|}{s^{\text{ref}}},$$

and η is the machine precision [8].

LM

The linear regression function LM was tested using the following straight-line model:

$$f(x, a_1, a_2) = a_1 + a_2 x,$$

with $\mathbf{a} = (a_1, a_2)$. This model is linear in its parameters **a**. Performance parameters for specifying reference data sets for this function include:

- the noise size,
- the location of the *x*-data, and
- the number *m* of points.

The performance measure $P(\mathbf{a})$ used to measure the departure of the computed regression coefficients **a** returned by the test software from the reference coefficients $\mathbf{a}^{\text{ref}}$ is given by

$$P(\mathbf{a}) = \log_{10}\left(1 + \frac{1}{\kappa(\mathbf{a})\eta}\frac{\left\|\mathbf{a} - \mathbf{a}^{\text{ref}}\right\|}{\left\|\mathbf{a}^{\text{ref}}\right\|}\right), \tag{9}$$

where $\kappa(\mathbf{a})$ measures the relative conditioning of the problem, and η is the machine precision [8]. Here, $\kappa(\mathbf{a})$ is computed as the condition number of the (column-normalised) Jacobian matrix *J* for the associated least-squares problem evaluated at the reference solution, i.e., the (i, j)th element of *J* is the derivative of the model *f* (straight-line) with respect to the *j*th regression parameter evaluated at the *i*th data point with $\mathbf{a} = \mathbf{a}^{\text{ref}}$. We also use a performance measure that relates to a *single* regression parameter *a* taken from the vector **a** of regression parameters that takes the form

$$P(a) = \log_{10}\left(1 + \frac{1}{\kappa(a)\eta}\frac{\left|a - a^{\text{ref}}\right|}{\left|a^{\text{ref}}\right|}\right), \tag{10}$$

where η is defined above and $\kappa(a)$ is the relative conditioning of the problem for determining the parameter *a*.

NLS

The nonlinear least-squares regression function NLS was tested using two different models as follows:

(a) Gaussian model

$$f(x, \mathbf{a}) = A\exp\left\{\frac{-(x - x_0)^2}{2\sigma^2}\right\},$$

with $\mathbf{a} = (A, x_0, \sigma)^{\text{T}}$. This model is nonlinear in its parameters **a**.

Performance parameters for specifying reference data sets for this function and model include:

- the noise size,
- the peak height *A*,
- the peak location x_0, and
- the peak width σ.

The performance measure $P(a)$ (see (10)) was used to measure the departure of each computed regression coefficient *a* returned by the test software from the reference coefficient a^{ref}.

(b) Sine-wave model

$$f(x, \mathbf{a}) = A\sin\{\omega x + \phi\},$$

where $\mathbf{a} = (A, \omega, \phi)^{\mathrm{T}}$. This model is nonlinear in its parameters **a**.

Performance parameters for specifying reference data sets for this function and model include:

- the noise size,
- the wave amplitude A,
- the wave velocity ω,
- the phase ϕ, and
- the number m of points.

The performance measure $P(a)$ (see (10)) was used to measure the departure of each computed regression coefficient a returned by the test software from the reference coefficient a^{ref}.

4.2 Mathematical and Trigonometric Functions

The following consistency tests were carried out:

T1. $\sin(\mathrm{asin}(x)) = x, -1 \le x \le 1.$

T2. $\mathrm{asin}(\sin(x + 2n\pi)) = x, -\pi/2 \le x \le +\pi/2.$

T3. $\cos(\mathrm{acos}(x)) = x, -1 \le x \le 1.$

T4. $\mathrm{acos}(\cos(x + 2n\pi)) = x, 0 \le x \le \pi.$

T5. $\tan(\mathrm{atan}(x)) = x.$

T6. $\mathrm{atan}(\tan(x + 2n\pi)) = x, -\pi/2 < x < \pi/2.$

T7. $\mathrm{atan}(y, x) = \mathrm{atan}(y/x), x \ne 0.$

T8. $\mathrm{atan}(\sin(x + 2n\pi), \cos(x + 2n\pi)) = x, -\pi < x \le +\pi.$

T9. $\sinh(\mathrm{asinh}(x)) = x.$

T10. $\mathrm{asinh}(\sinh(x)) = x.$

T11. $\cosh(\mathrm{acosh}(x)) = x, x \ge 1.$

T12. $\mathrm{acosh}(\cosh(x)) = x.$

T13. $\tanh(\mathrm{atanh}(x)) = x, -1 < x < +1.$

T14. $\mathrm{atanh}(\tanh(x)) = x.$

T15. $\sin^2(x) + \cos^2(x) = 1.$

T16. $\tan(x) = \sin(x)/\cos(x).$

T17. $\sinh(x) = (\exp(x) - \exp(-x))/2.$

T18. $\cosh(x) = (\exp(x) + \exp(-x))/2.$

T19. $\tanh(x) = \sinh(x)/\cosh(x).$

T20. $\exp(\log(x)) = x.$

T21. $\log(\exp(x)) = x.$

T22. $\log10(x) = \log(x)/\log(10).$

T23. $\log(x, \exp(1)) = \log(x)$.

T24. $\log(x, 10) = \log10(x)$.

T25. $\log(y^x, y) = x$.

In each case, the performance measure $P(x)$ given by (6),

$$P(x) = \log_{10}\left(1 + \frac{d_R(x)}{\eta}\right),$$

is used to make judgements about the test software in the following ways:

1. By highlighting cases for which the value of the performance measure is significantly greater than zero (equivalently, cases for which the relative measure $d_R(x)$ of consistency is significantly greater than η).
2. By identifying discontinuities in value and/or slope of the performance measure as a function of its argument, e.g., performance parameter.
3. By noting trends in the values of the performance measure, e.g., periodic behaviour of the performance measure.

4.3 Statistical Distributions

The results returned by individual functions are compared with those returned by the corresponding functions from the IMSL Fortran 90 Math Library (version 3.0) provided with the Digital Visual Fortran (version 5.0D) software package [14]. Tables of reference values were generated and imported into the S-PLUS environment where direct comparisons could be made.

In each case, the performance measure $P(x)$ given by (6),

$$P(x) = \log_{10}\left(1 + \frac{d_R(x)}{\eta}\right),$$

is used to make judgements about the test software in the ways described in Section 4.2.

5. Presentation and Interpretation of Results

5.1 Regression Functions

We give here some typical results obtained from the testing of the S-PLUS regression functions whose specifications are given in Section 3.1. In Section 4.1 the performance parameters and performance measures for each regression function are listed. The performance measures are presented graphically as functions of the performance parameters in the figures contained in Appendix A.

MEAN and **VAR**

Figures 1–6 show plots of the performance measures given by (7) and (8) for the sample mean and sample standard deviation, respectively, against the following performance parameters:

- the number of sample data points,
- the reference sample mean, and
- the reference sample standard deviation.

Nominal values for these performance measures are given in Table 1.

From Figures 1–3 it can be seen that there is no significant degradation in the performance of the function MEAN for the ranges of the performance parameters over which the function was tested. In all three graphs, the largest value of the performance measure is less than unity, indicating that the accuracy obtained from this function is very close to that for an optimal algorithm to evaluate the sample mean.

The results for the sample standard deviation, calculated using the function VAR, are shown in Figures 4–6. As for the case of the function MEAN, there is no observable degradation of the performance of the function for the ranges of the performance parameters over which the function was tested. Again, the largest value of the performance measure is less than unity in all three graphs. These results imply that even in fairly extreme cases only at most one significant figure of accuracy is lost over and above that which would be obtained from an optimal algorithm.

The results presented here lead us to conclude that a numerically reliable method for the calculation of the sample standard deviation has been implemented in S-PLUS. These compare with the results obtained from the testing of the equivalent function within the Microsoft Excel spreadsheet package [10], which suggests that the Excel function implements a *numerically* unreliable, though *mathematically* correct, formula. Whereas *pre-processing* of the sample values [15] prior to application of the Excel function is necessary in some circumstances to obtain accurate results, this need not be done prior to application of the S-PLUS function.

Parameter	*Nominal value*
reference sample mean	+1.437575400258079
number of points	50
reference sample standard deviation	0.1

Table 1 Nominal values for the performance parameters for the testing of the MEAN and VAR worksheet functions.

LM

Figures 7–14 show performance profiles obtained from the testing of the function LM using a straight-line model. The performance measure plotted is that defined by equation (10) applied to each model parameter (the intercept a_1 and the slope a_2, respectively), and the performance parameters are:

- the noise size,
- the location of the data *x*-values, and
- the number of points.

Nominal values for the performance parameters are given in Table 2.

Parameter	*Nominal value*
noise size *ns*	5.0
location (median) of data *x*-values	2.1
number of points	421

Table 2 Nominal values for the performance parameters for the testing of the LM function.

A slight degradation in the performance of the function is observed when the noise size is increased to comparatively large values (Figures 7–10)[12]. However, for neither model parameter are the results particularly significant as the performance measure attains largest values of only approximately 2.5 over the range of noise values considered, i.e., at worst, between two and three significant figures of accuracy are lost compared to a reference algorithm.

No significant change in the performance of the function is noted as the location of the data *x*-values is varied (parts of the performance profiles are shown in Figures 11 and 12). The results are to be compared with those for the equivalent functions in Excel [10] and MathCAD [11] where some degradation in performance as a function of this performance parameter was observed.

Finally, there is essentially no degradation in the performance of the function as the number of sample data points is increased (Figures 13 and 14).

NLS

Figures 15–18 and 19–23 show performance profiles obtained from the testing of function NLS using, respectively, the Gaussian and sine-wave models. The performance measure plotted is that defined by equation (10) applied to each model parameter. Nominal values for the performance parameters for the testing using the two models are given in Tables 3 and 4, respectively. The models were chosen as they present moderately difficult non-linear regression problems, and are encountered in a number of metrology applications.

For testing using the Gaussian model, the behaviour of the performance measures as functions of the peak amplitude A, peak location x_0 and peak width σ (Figures 16–18) is essentially flat showing that there is no degradation in the performance of the function as these parameters are varied. The fact that the performance measure assumes a *consistent* value suggests that the observed accuracy of the results may be due to truncation errors arising from the convergence criterion used to terminate an iterative algorithm implemented by the function.

For testing using the sine-wave model, the behaviour of the performance measures as functions of the wave amplitude A, wave velocity ω and phase angle ϕ (Figures 20–22) is again essentially flat, although there is some degradation in performance for small amplitudes and phase angles close to zero and π. As before, we expect the observed accuracy of the results to reflect a convergence criterion. The results presented in [11] for the equivalent MathCAD function using this model indicated that for some data sets convergence to a (local) solution that was different from the reference solution occurred: this behaviour is not observed in the results for S-PLUS presented here.

For testing using both models (and particularly for the sine-wave model), there is some degradation in the performance of the function as the noise size is increased (Figures 15 and 19) and the number of points is reduced (Figure 23). Generally, we would expect the regression problem to be more difficult, i.e., to have a larger conditioning, when the solution is less well defined by the data. This can happen when there is a greater amount of data noise or fewer data points. The observed degradation for the ranges of data sets considered implies the performance of the function is slightly, but not significantly, worse than would be expected accounting for the change in conditioning.

[12] Figures 7 and 8 provide detail for the results shown in Figures 9 and 10. For the latter, the performance parameter (noise size) is presented on a logarithmic scale.

Parameter	*Nominal value*
peak height A	1.0
peak location x_0	1000
peak width σ	1.0
noise size *ns*	0.001
location (median) of data *x*-values	1000

Table 3 Nominal values for the performance parameters for the testing of the NLS function using a Gaussian model function.

Parameter	*Nominal value*
wave amplitude A	1.0
wave velocity ω	1.0
phase angle ϕ	$\pi/2$
noise size *ns*	0.001
number of points	421

Table 4 Nominal values for the performance parameters for the testing of the NLS function using a sine-wave model function.

5.2 Mathematical and Trigonometric Functions

Figures 24–36 in Appendix B show plots of the performance measure (6) for a selection of the consistency checks described in Section 4.2 for the mathematical and trigonometric functions. We have only displayed those results that are significant (in terms of the value of the performance measure). Note that some care is needed in interpreting these performance profiles in the neighbourhood of the origin, since the measure (6) is based on the *relative* departure between the test and reference results. We note that:

- For many of the profiles, including T1 (Figure 24), T3 (Figure 26) and T5 (Figure 28), there are isolated points (*x*-values) at which the performance measure is zero interspersed with points for which the performance measure is non-zero (but generally small). There is no significance since this effect is almost a random one.
- For some of the profiles, such as T11 (Figure 30), T13 (Figure 32) and T20 (Figure 36), (interlaced) subsets of the results appear to exhibit systematic behaviour. The systematic behaviour might be explained by the fact that an innate sensitivity associated with the consistency checks is not being fully accounted for in the performance measure. For some input values the floating point numbers involved are such that the consistency check is exact; for other values the problem sensitivity will indicate an inflation of the computational precision η that manifests itself as a non-zero performance value. This effect is a consequence of the nature of floating point arithmetic, and may be significant depending on the intended use of the functions. It is the inflation of η that is observed in the graphs presented.
- For T14 (Figure 33) the value of the performance measure becomes large towards the end of the interval of *x* over which the consistency check is applied. (It should be noted that the use of a "zero of a function" technique for evaluating the inverse of a function, e.g., the use of "tanh" and a bisection algorithm for evaluating "atanh", can be expected to yield the maximum possible precision for this consistency check.)

- The results for T3 (Figure 26), T5 (Figure 28) and T14 (Figure 33) are similar in appearance to those obtained by implementing the same consistency checks using MathCAD [11].

5.3 Statistical Distributions

Figures 37–40 in Appendix C show plots of the performance measure (6) for the comparison of a selection of the S-PLUS statistical distributions against the equivalent IMSL functions. We show the results for the same distributions as reported in the testing of MathCAD [11]. We note that:

- The departure between equivalent functions for the beta distribution (Figure 37) does *not* exhibit systematic behaviour or discontinuities. This compares with the results reported for the Excel [10] and MathCAD [11] functions for the beta distribution for which the performance measure showed similar (systematic) behaviour.
- Since for many applications in metrology only a few significant figures of accuracy are required when evaluating these distributions, the results reported here are expected to be sufficiently small.[13]

6. Conclusions

In this report we have described the application of a general methodology [8] for testing the numerical accuracy of scientific software to specific functions taken from the S-PLUS statistical software package. Each stage of the methodology, from documenting a specification for each function tested through the definition of performance parameters and measures to the presentation and interpretation of the test results, has been described. In this way, and by stating any assumptions made in the application of the methodology, the testing undertaken is made as objective as possible given the nature of the testing.

It has not been the intention in this work to consider *all* the intrinsic functions included with S-PLUS, but to concentrate on specific functions that are relevant to the numerical processing of measurement data undertaken in metrology applications. The work has, therefore, concentrated on regression functions, mathematical and trigonometric functions, and statistical distributions (including their inverses) that underpin the requirements of metrology. Consequently, conclusions drawn from the testing undertaken of a particular function must be interpreted in the context of that function only, and not in the context of other functions or the S-PLUS statistical software package as a whole. Furthermore, the tests described here have been carried out in such a way that the functions have been used without taking account of information elsewhere, e.g., as contained in publications on S-PLUS or information posted on the World Wide Web; only the documentation available on-line as part of the normal S-PLUS software "environment" was used. This mode of use is deliberate, since we believe it accords with that adopted by most users generally and within metrology in particular.

The test results are intended primarily to help users understand whether for a particular application the functions used are fit for purpose, and to understand the limitations (if any) of those functions. Where the testing had indicated ways in which a user may make better use of a given function, for example by pre-processing the data prior to its application, to overcome any such limitation, this information can be provided. However in the case of the S-PLUS functions this does not appear to be necessary.

[13] However, note that deciding fitness for purpose (in terms of numerical accuracy of results) needs to account for the way the results returned by the software are subsequently used. Some of the issues that affect how users decide whether software is fit for purpose within the context of a metrology application are considered in [10, 15].

Some of the highlights of the test results reported in this work are as follows:

- The functions MEAN and VAR return results to an accuracy that is very close to that for numerically optimal algorithms for the sample mean and standard deviation calculations.
- The performance of the linear regression function is generally good. There does not appear to be a requirement for pre-processing of the input data that has been recommended following similar tests conducted with equivalent functions in Excel [10] and MathCAD [11].
- The performance of the nonlinear regression function for the two nonlinear models considered (Gaussian and sine-wave) is generally good. However, where the amount of noise is large or there are few points, slight degradation beyond that which is accounted for by problem conditioning is observed. For all data sets considered, convergence to the reference solution rather than some other (local) solution was obtained. It can be expected that the observed accuracy of the results is a reflection of the termination criterion employed by the algorithm implemented.
- The results of consistency checks for the mathematical and trigonometric functions are generally good: a small number of examples of systematic behaviour in the values of the performance measure for these checks have been noted. The systematic behaviour may be due to the sensitivity of the consistency checks not being accounted for in the performance measures, and further work will be necessary to address this issue.
- The results of the comparison of the statistical distributions with other equivalent software are generally good: no systematic behaviour in the values of the performance measures was observed.

7. Acknowledgements

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

We would also like to acknowledge Professor M. G. Cox and Dr P. M. Harris for their considerable input into this report by way of supervision and reviewing of the work.

8. References

[1] D Rayner. *Initial report on status of software and mathematics in each metrology area*. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CISE 18/99, February 1999.

[2] M G Cox, M P Dainton, P M Harris and B A Wichmann. *Survey report on 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 22/99, March 1999.

[3] B.D. McCullough. Assessing the reliability of statistical software: Part I. *The American Statistician*, **52**(4), pp 358-366, 1998.

[4] B.D. McCullough. Assessing the reliability of statistical software: Part II. *The American Statistician*, **53**(2), pp 149-159, 1999.

[5] S.L.R. Ellison, M.G. Cox, A.B. Forbes, B.P. Butler, S.A. Hannaby, P.M. Harris, and S.M. Hodson. Development of data sets for the validation of analytical instrumentation. *Journal*

of AOAC International, **77**(3), pp 1-5, 1994.

[6] Statistics Software Qualification. Edited by B.P. Butler, M.G. Cox, S.L.R. Ellison and W.A. Hardcastle. The Royal Society of Chemistry, 1996.

[7] 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.

[8] 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.

[9] H.R. Cook, M.G. Cox, M.P. Dainton and P.M. Harris. *Testing spreadsheets and other packages used in metrology: A case study*. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CISE 26/99, September 1999.

[10] 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.

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

[12] S-Plus 4.0 Release 3 for Windows Copyright ©1988-1997, MathSoft Inc.

[13] Matlab Version 5.3.0.10183 (R11). Copyright ©1984-1999, The MathsWork Inc.

[14] IMSL Fortran 90 Math/Library (version 3.0) provided with Digital Visual Fortran (version 5.0.D, Professional Edition), Digital Equipment Corporation.

[15] M.G. Cox and P.M. Harris. *Guidelines to help users select and use software for their metrology applications*. Report to the National Measurement System Policy Unit, Department of Trade and Industry, from the UK Software Support for Metrology Programme. NPL Report CISE 04/00, September 2000.

Appendix A: Results for Regression Functions

MEAN

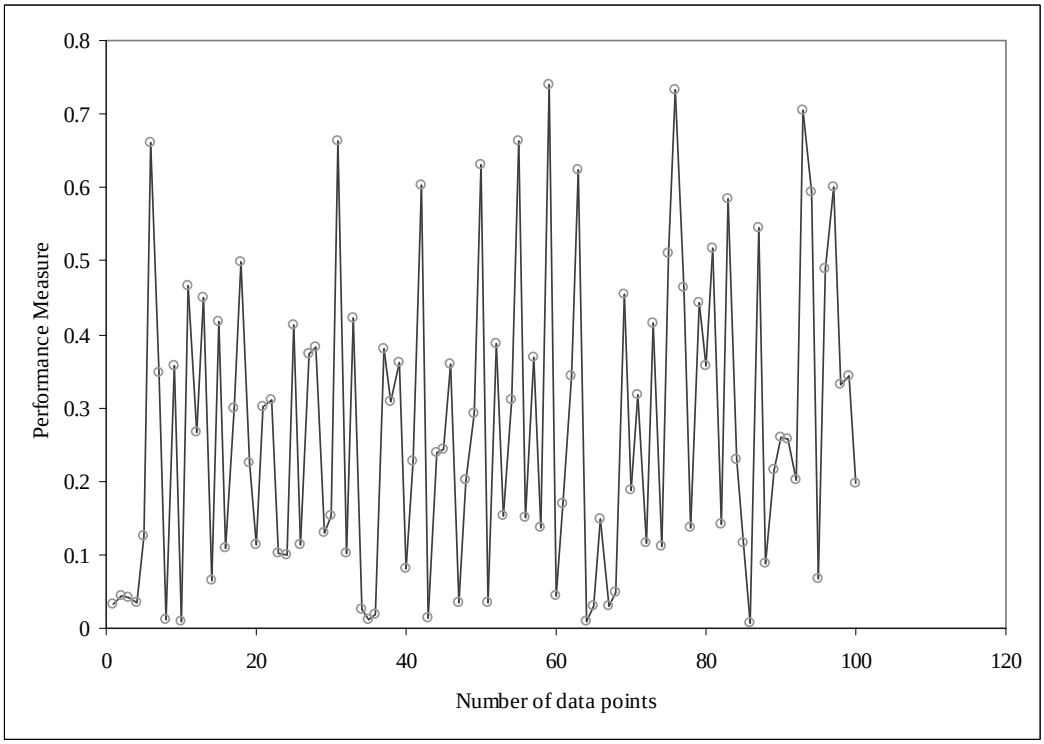


Figure 1: Plot of the performance measure $P(\bar{x})$ for the sample mean $\bar{x}$ against the number of data points.

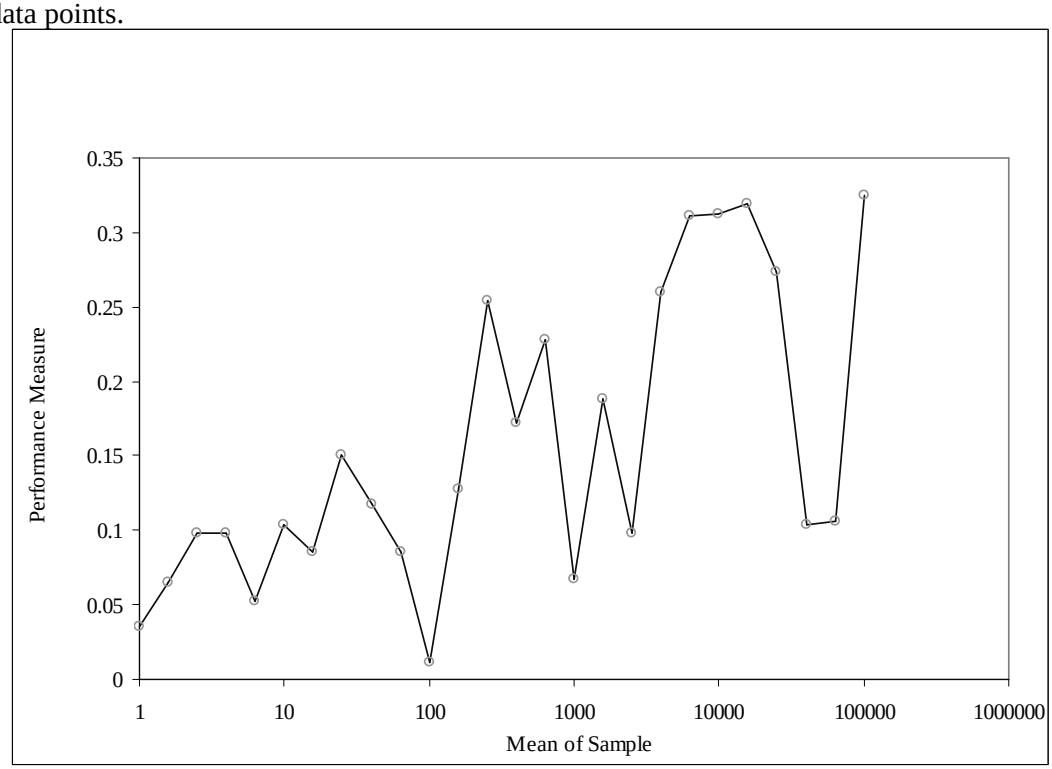


Figure 2: Plot of the performance measure $P(\bar{x})$ for the sample mean $\bar{x}$ against the reference sample mean.

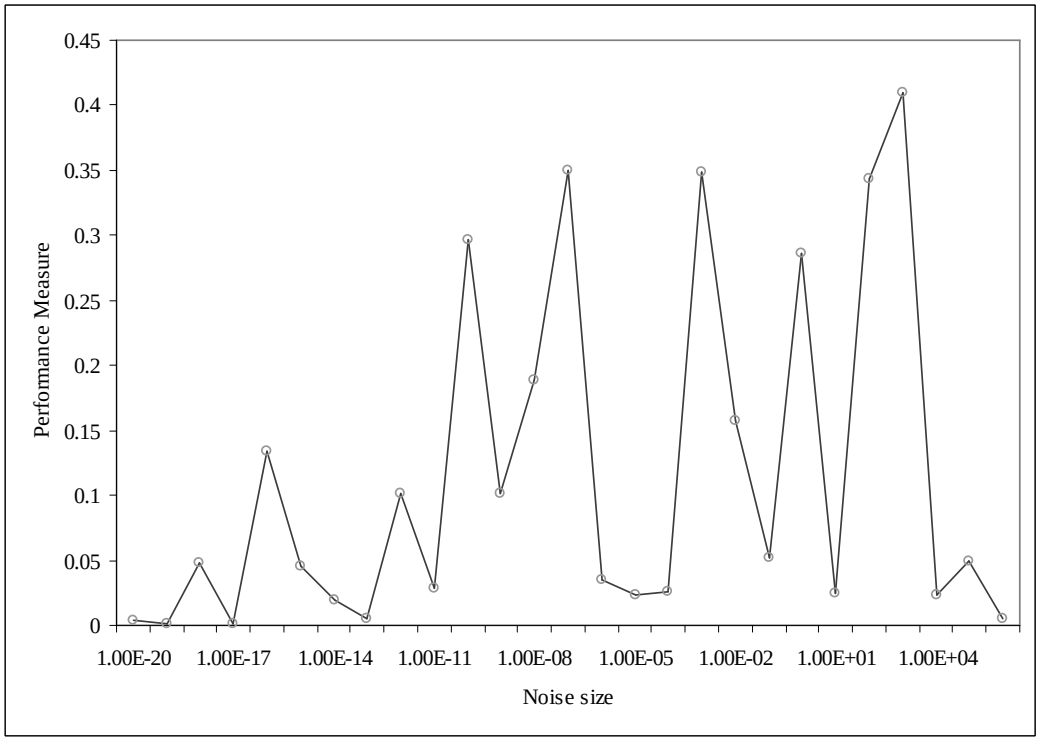


Figure 3: Plot of the performance measure $P(\bar{x})$ for the sample mean $\bar{x}$ against the reference sample standard deviation.

VAR

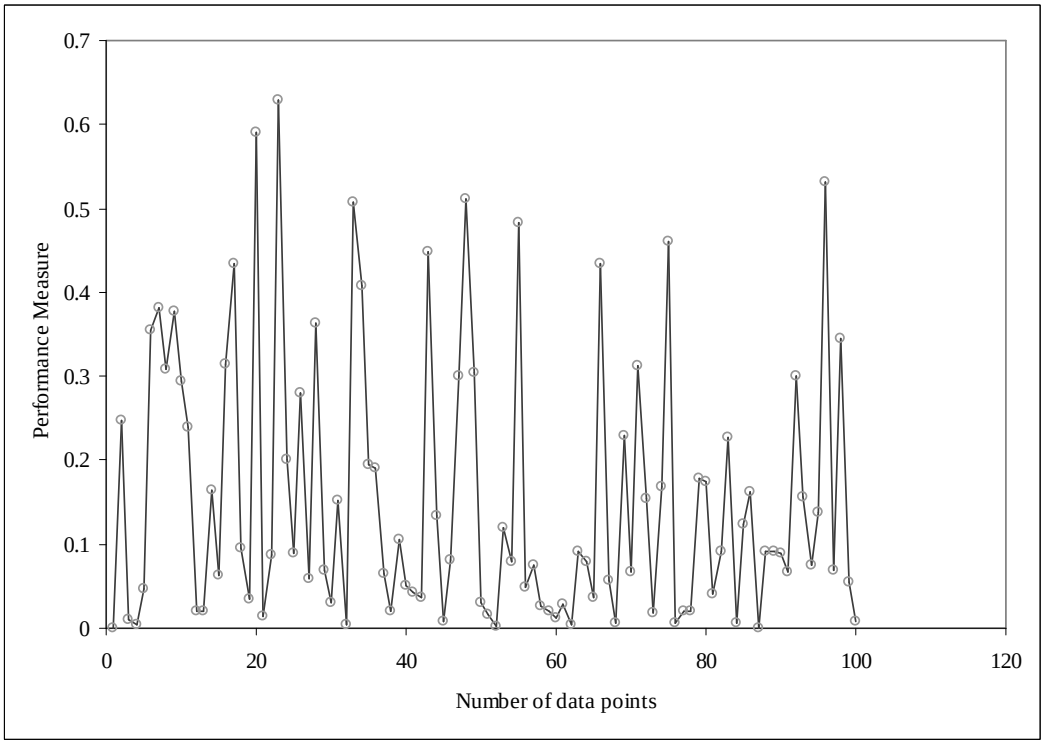


Figure 4: Plot of the performance measure $P(s)$ for the sample standard deviation s against the number of data points.

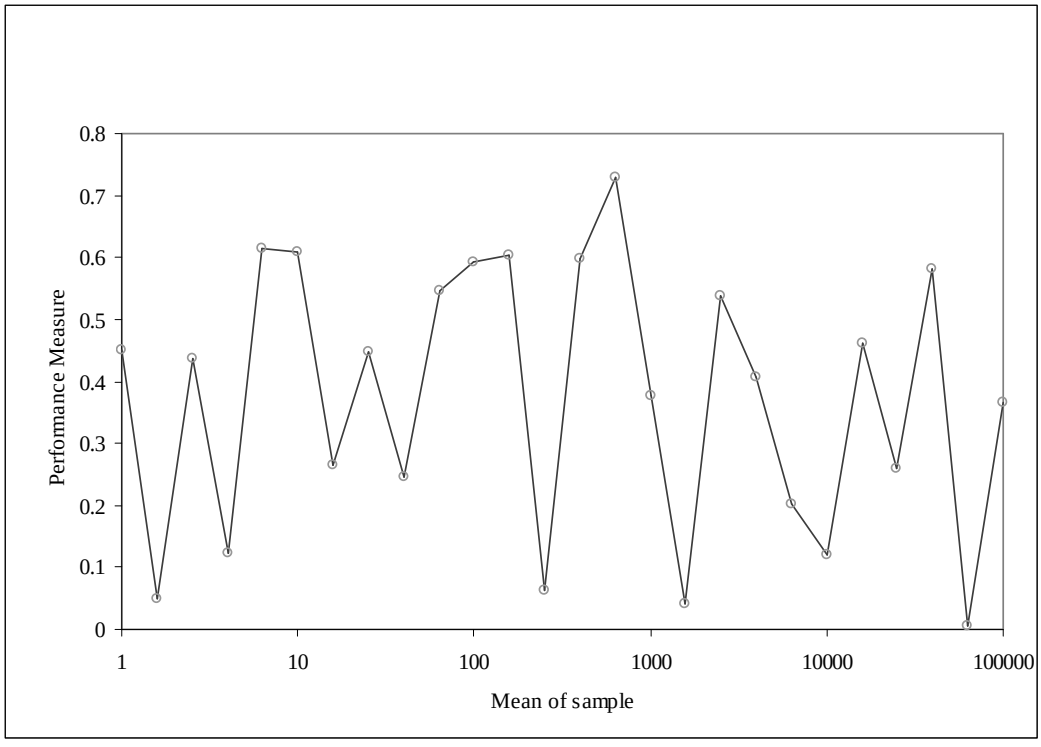


Figure 5: Plot of the performance measure $P(s)$ for the sample standard deviation s against the reference sample mean.

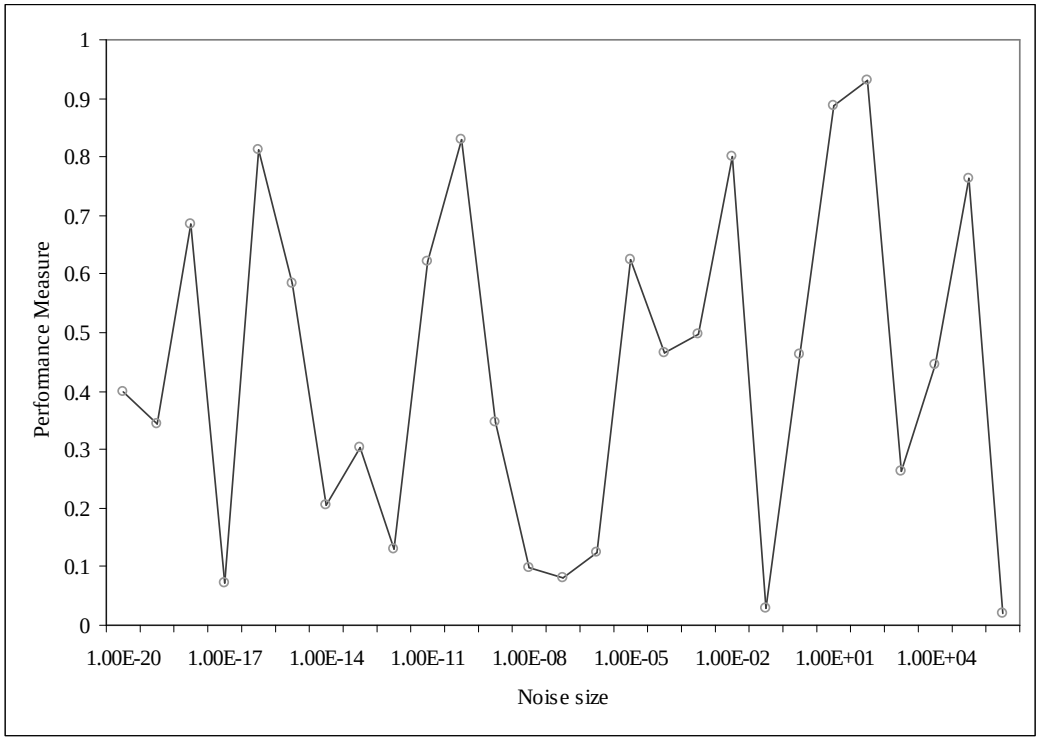


Figure 6: Plot of the performance measure $P(s)$ for the sample standard deviation s against the reference sample standard deviation.

LM

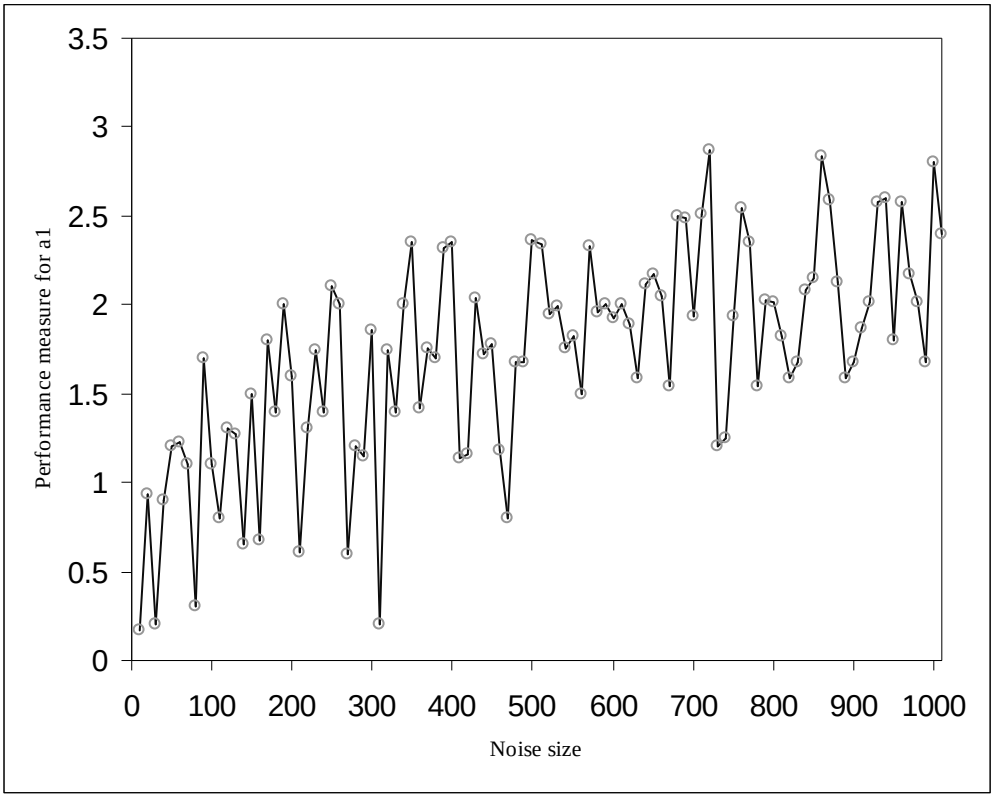


Figure 7: Plot of the performance measure $P(a_1)$ against noise size.

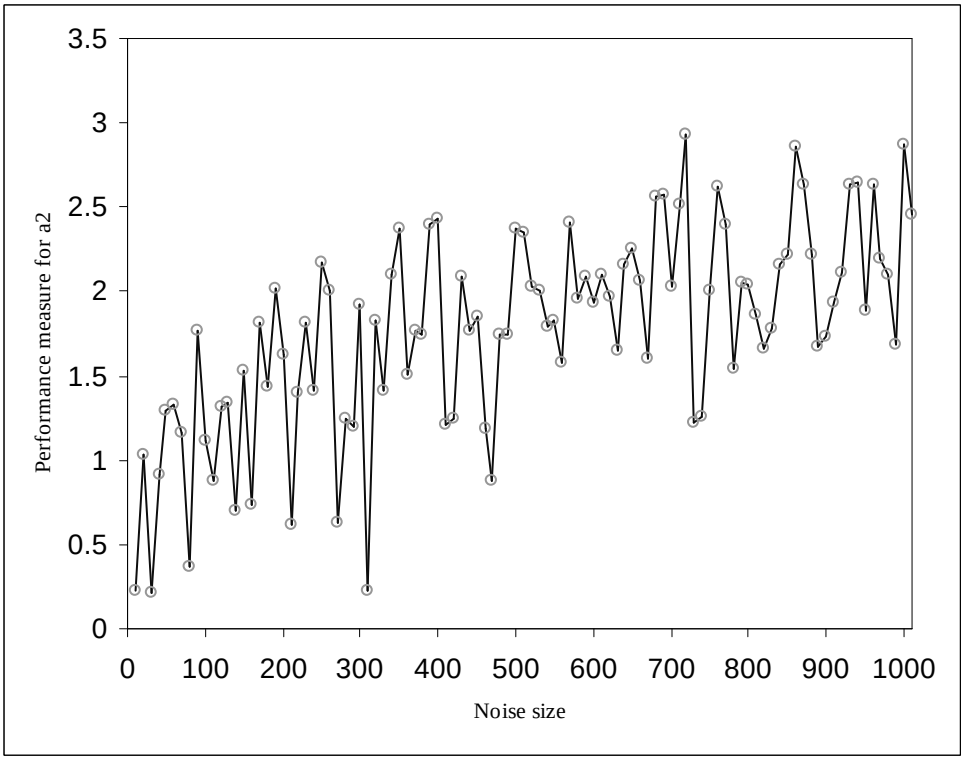


Figure 8: Plot of the performance measure $P(a_2)$ against noise size.

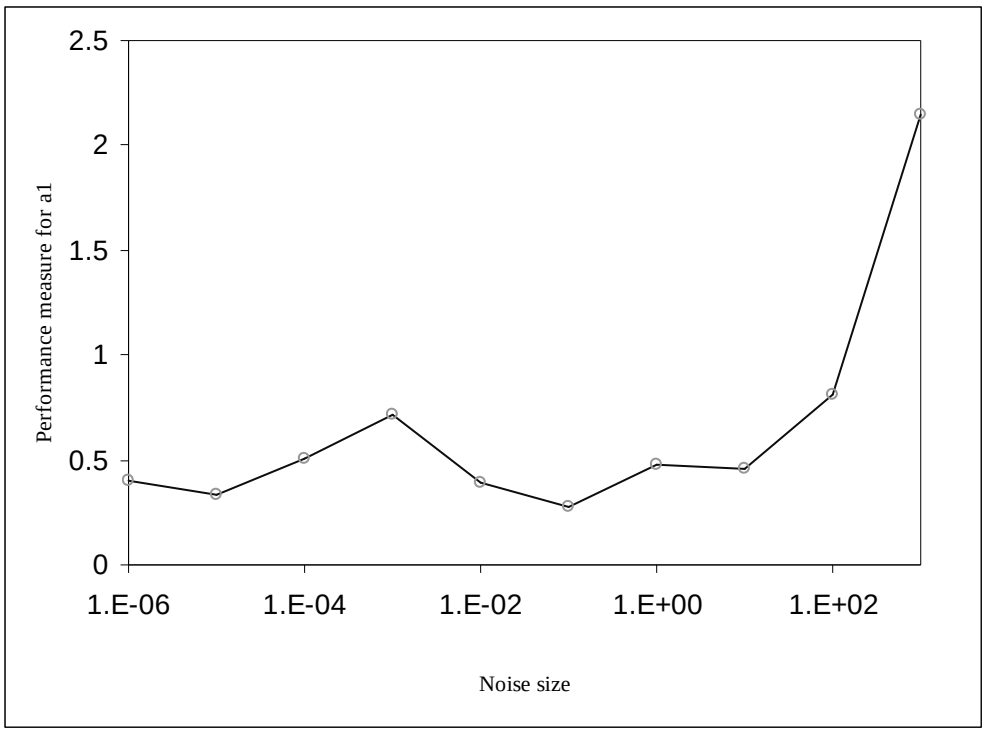


Figure 9: Plot of the performance measure $P(a_1)$ against noise size.

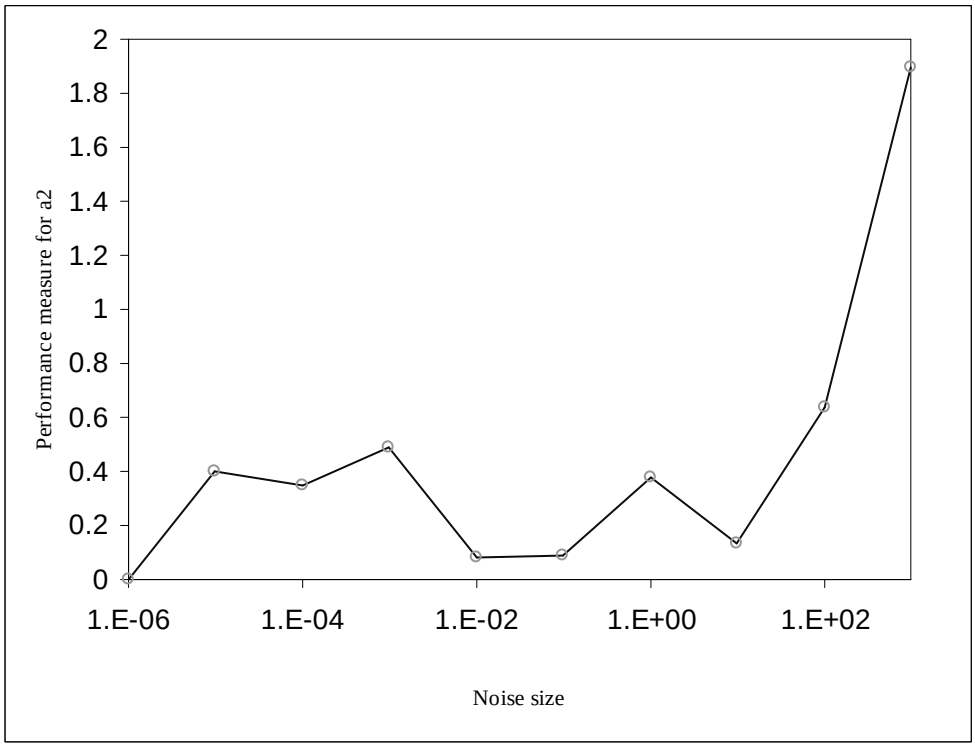


Figure 10: Plot of the performance measure $P(a_2)$ against noise size.

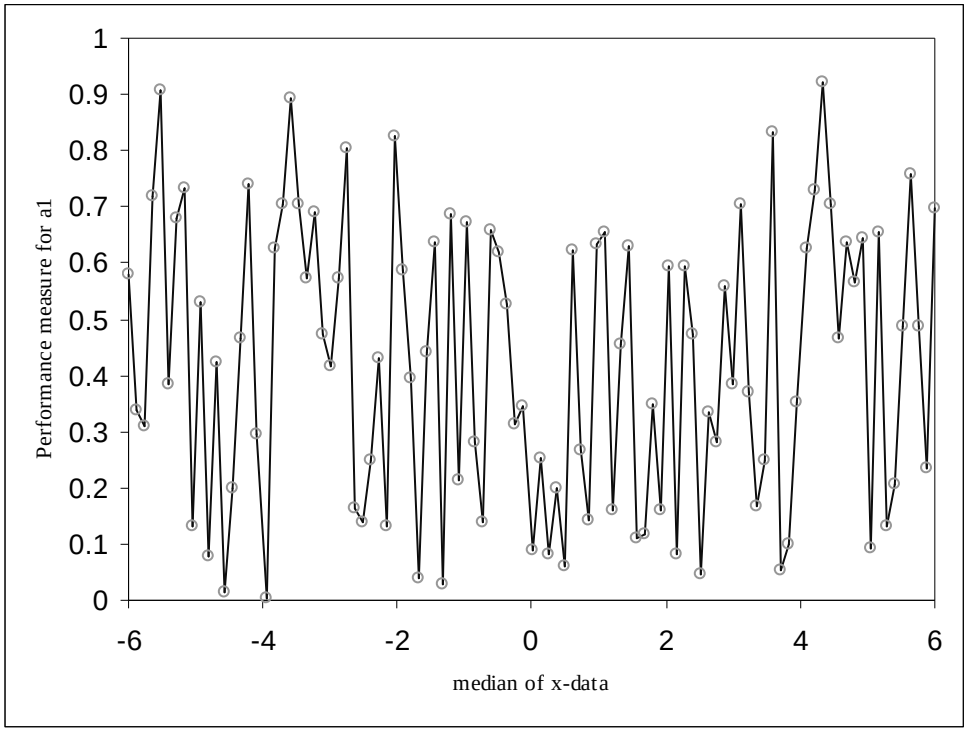


Figure 11: Plot of the performance measure $P(a_1)$ against data location.

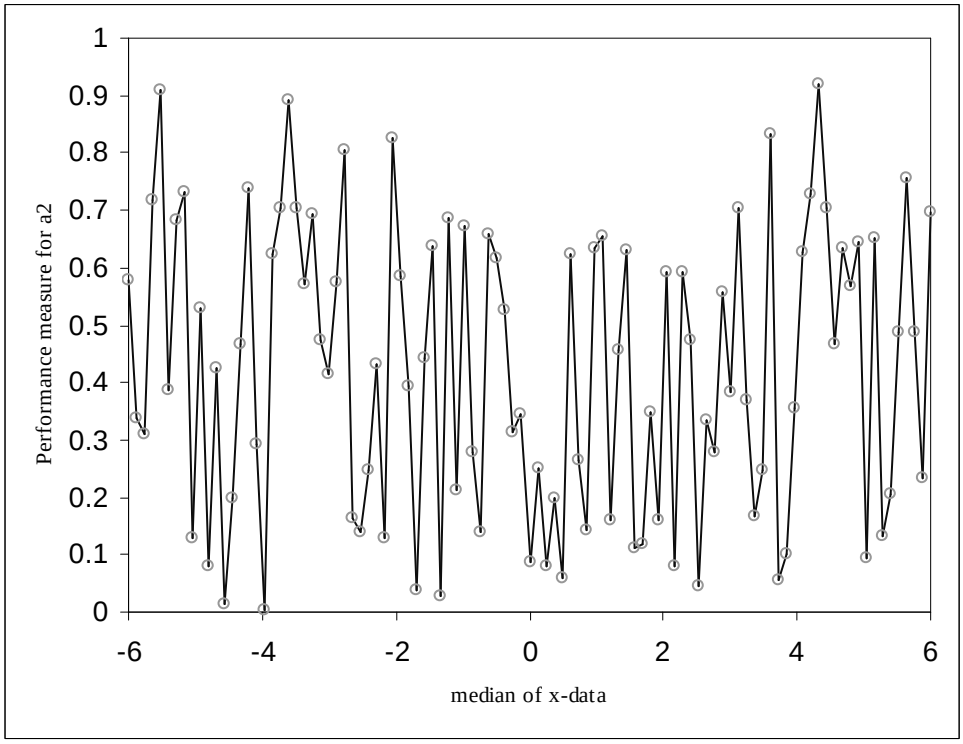


Figure 12: Plot of the performance measure $P(a_2)$ against data location.

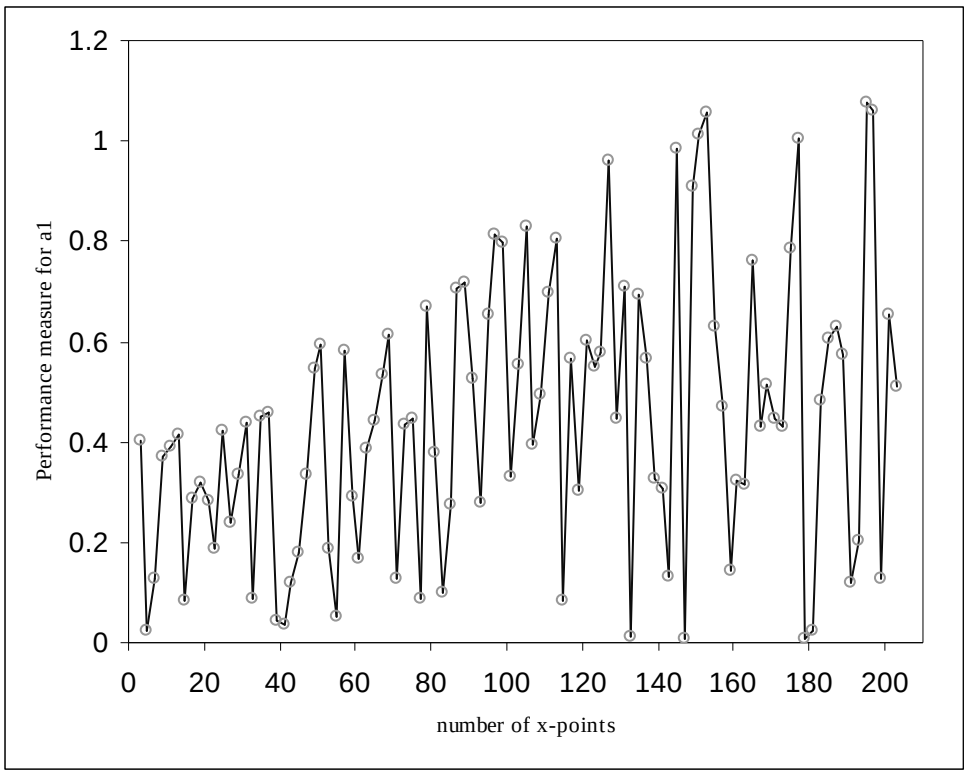


Figure 13: Plot of the performance measure $P(a_1)$ against number of data points.

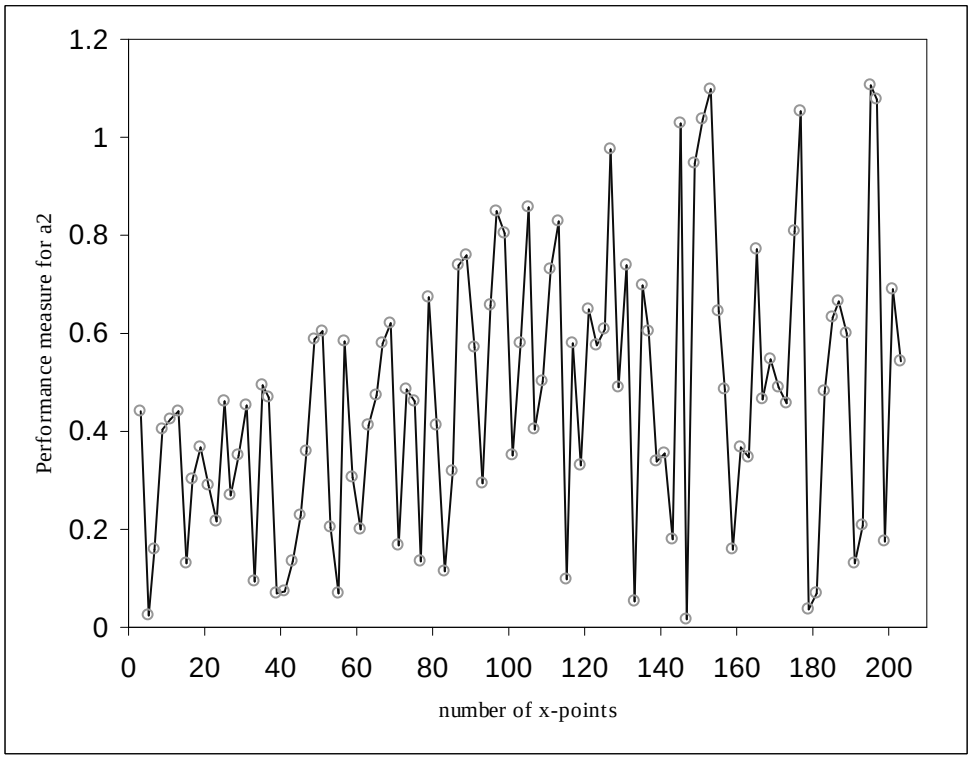


Figure 14: Plot of the performance measure $P(a_2)$ against number of data points.

NLS: Gaussian Model

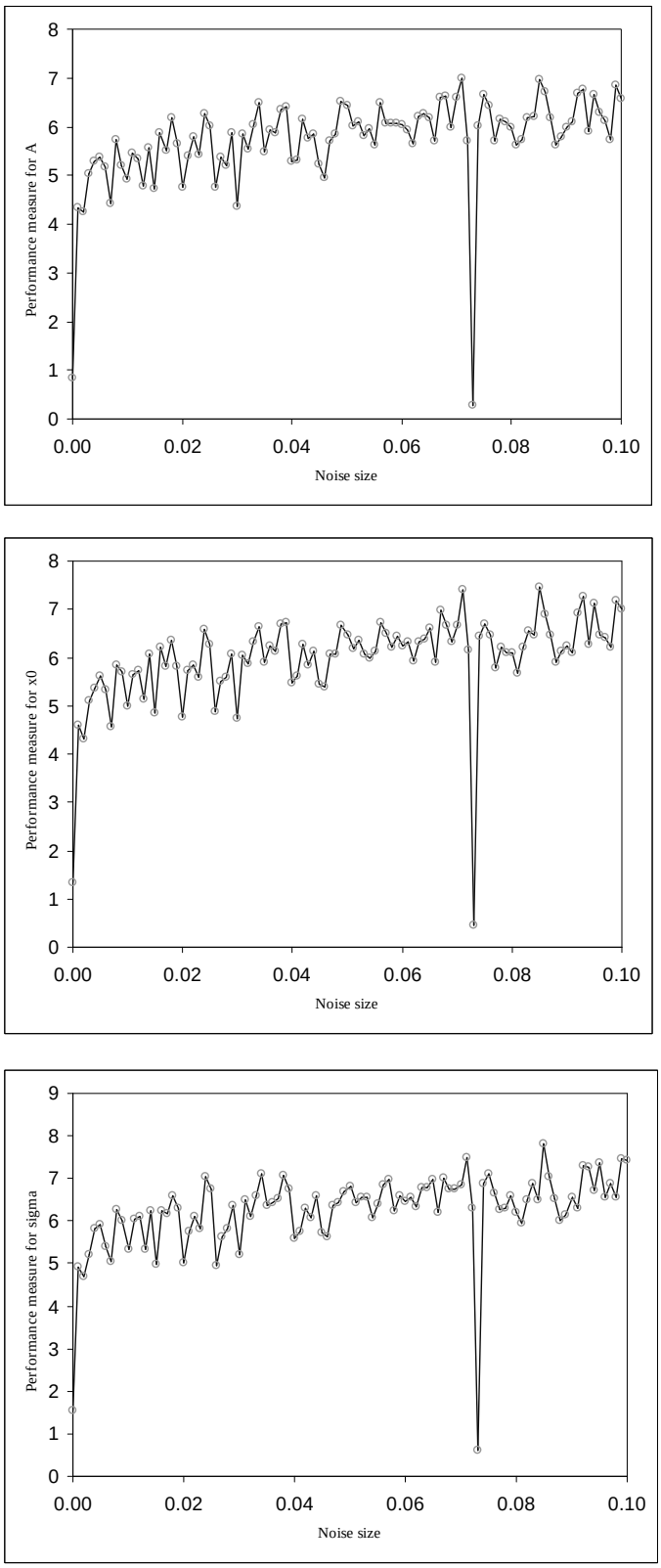


Figure 15: Plots of the performance measures $P(A)$, $P(x_0)$ and $P(\sigma)$ against noise size.

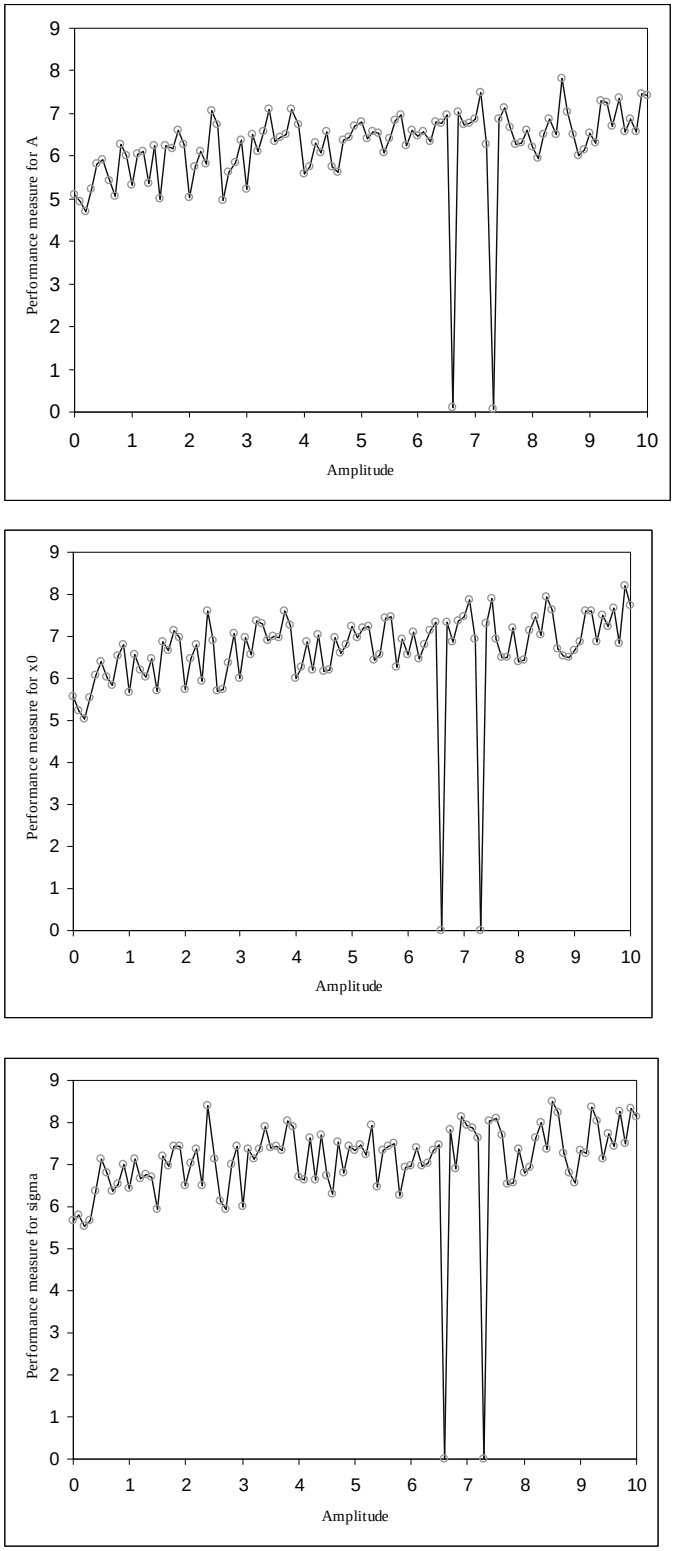


Figure 16: Plots of the performance measures $P(A)$, $P(x_0)$ and $P(\sigma)$ against amplitude.

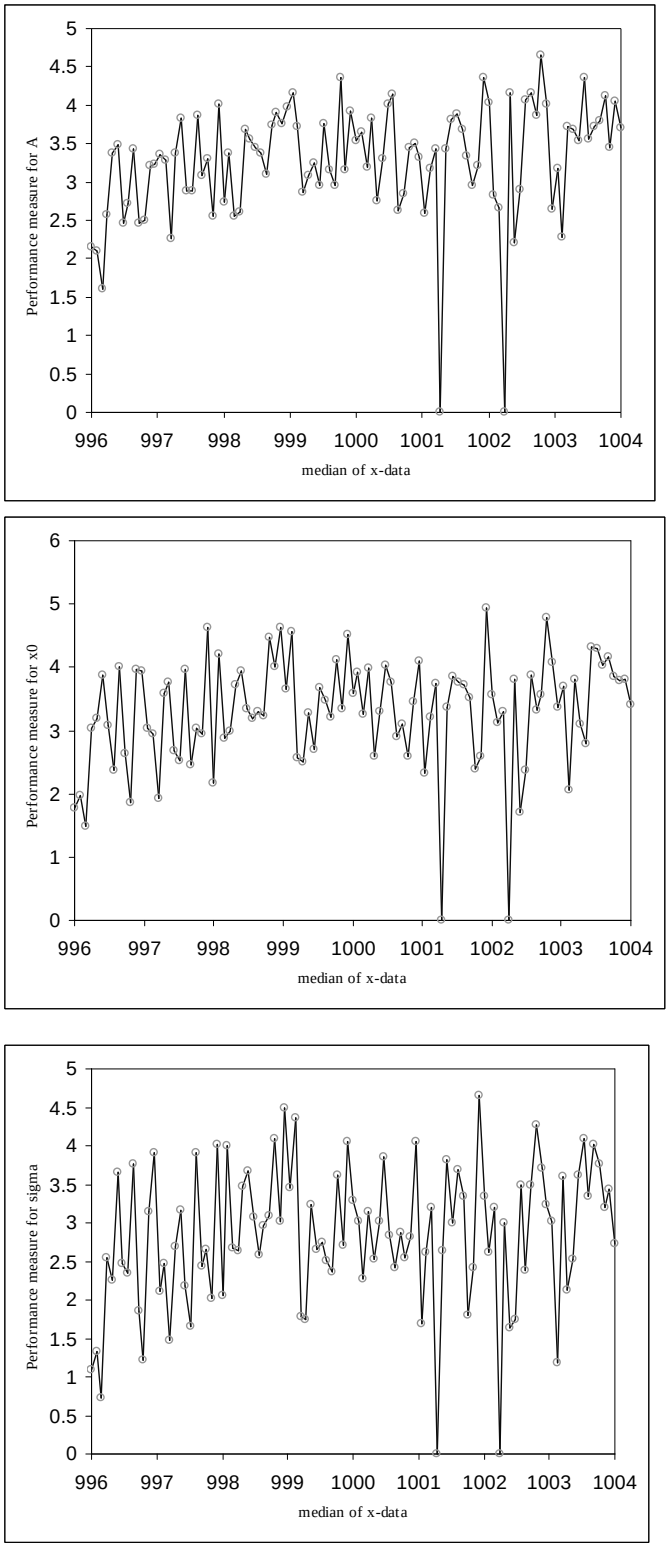


Figure 17: Plots of the performance measures $P(A)$, $P(x_0)$ and $P(\sigma)$ against peak location.

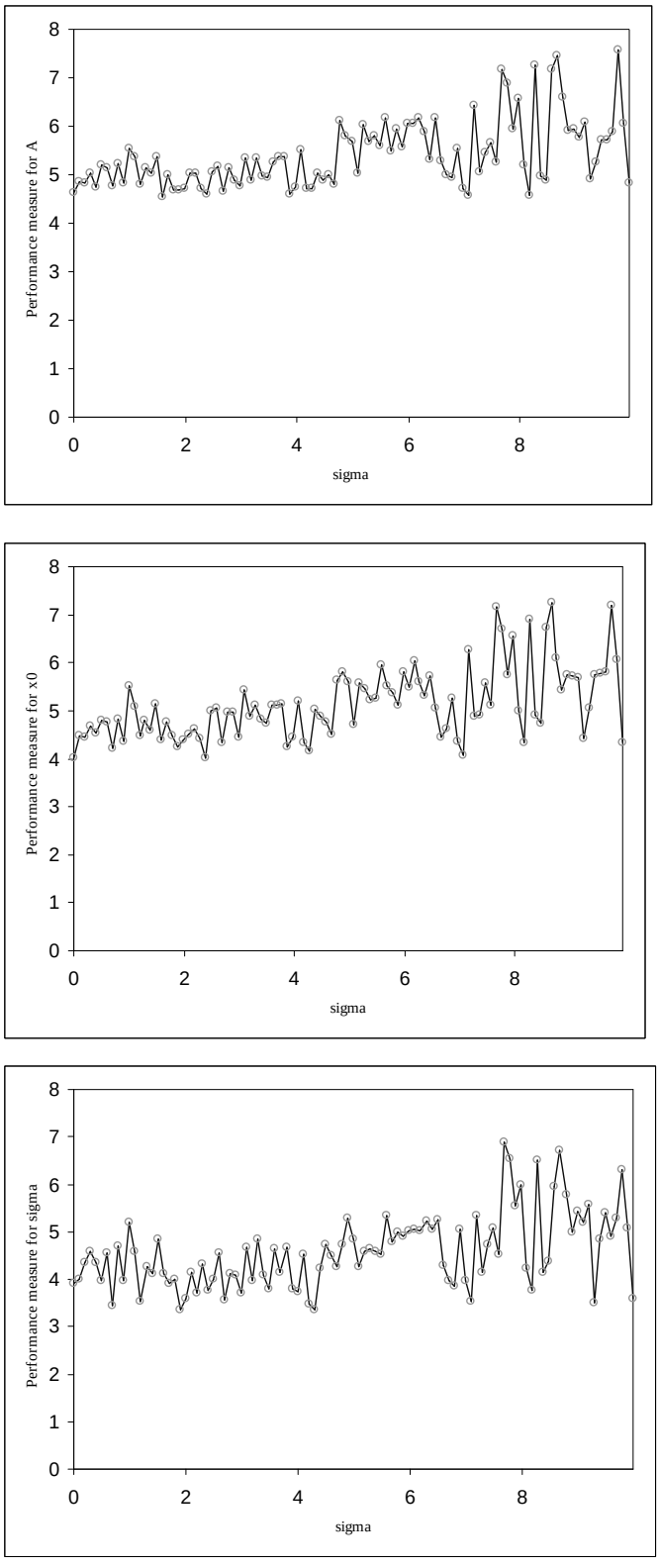
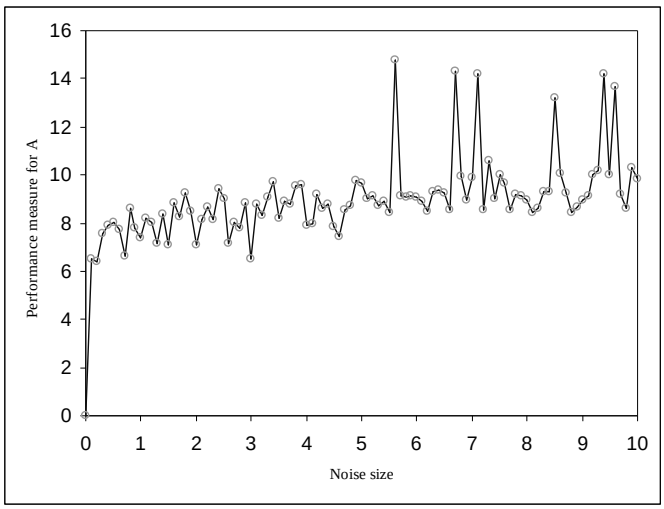
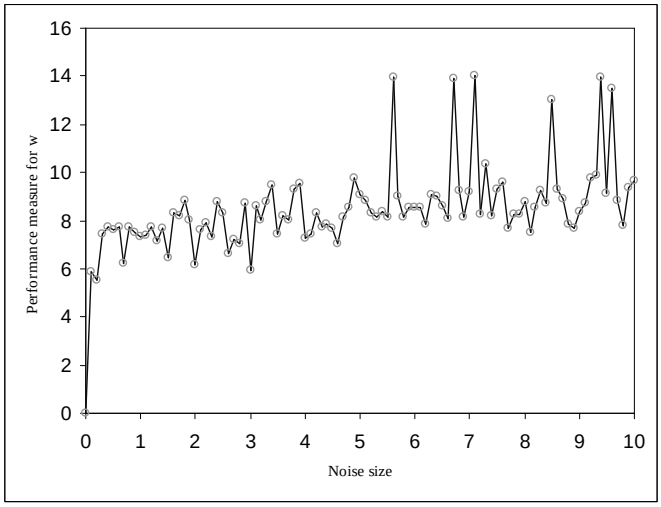


Figure 18: Plots of the performance measures $P(A)$, $P(x_0)$ and $P(\sigma)$ against peak width.

NLS: Sine-wave model





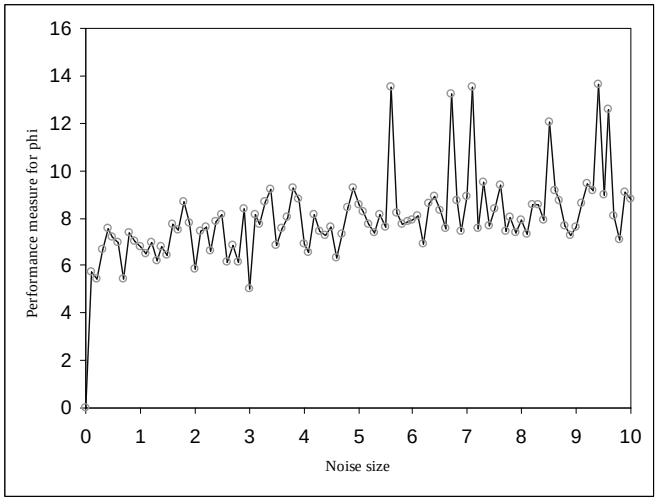


Figure 19: Plots of the performance measures $P(A)$, $P(\omega)$ and $P(\phi)$ against noise size.

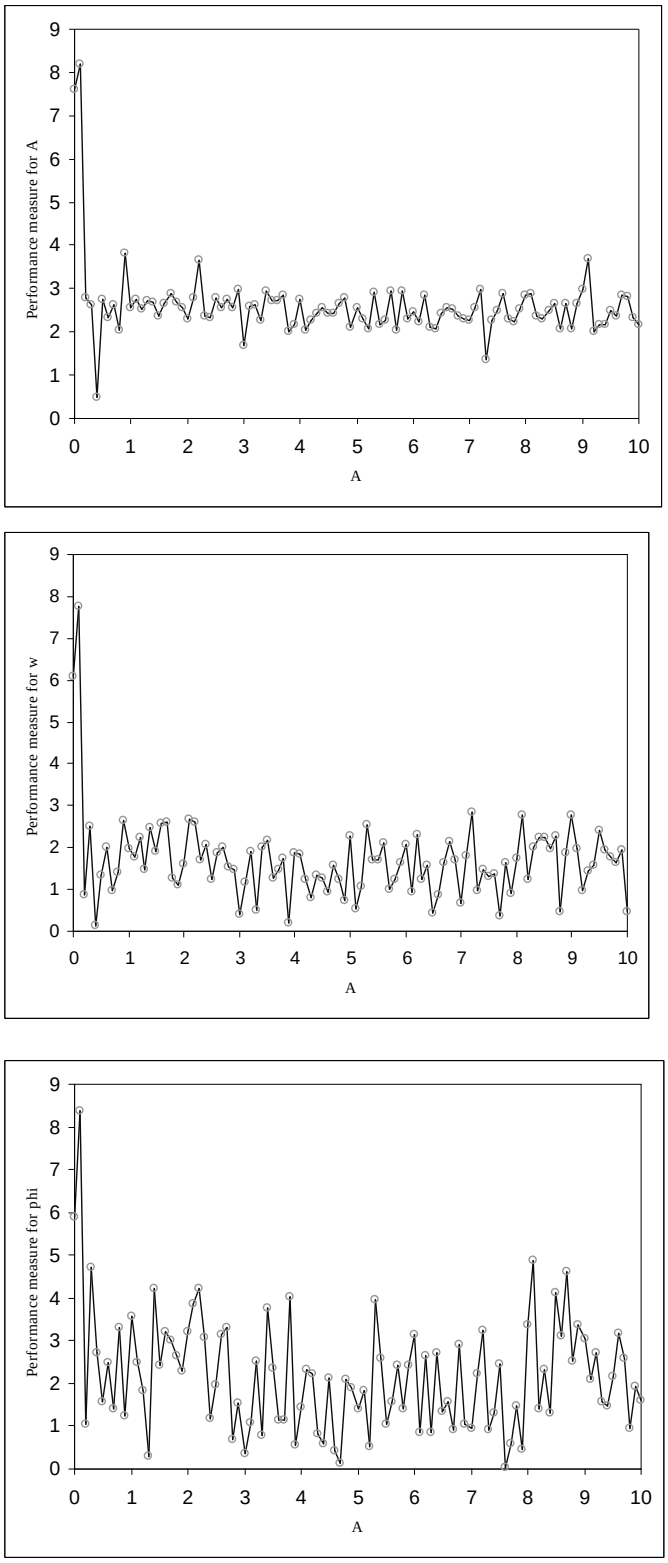


Figure 20: Plots of the performance measures $P(A)$, $P(\omega)$ and $P(\phi)$ against amplitude.

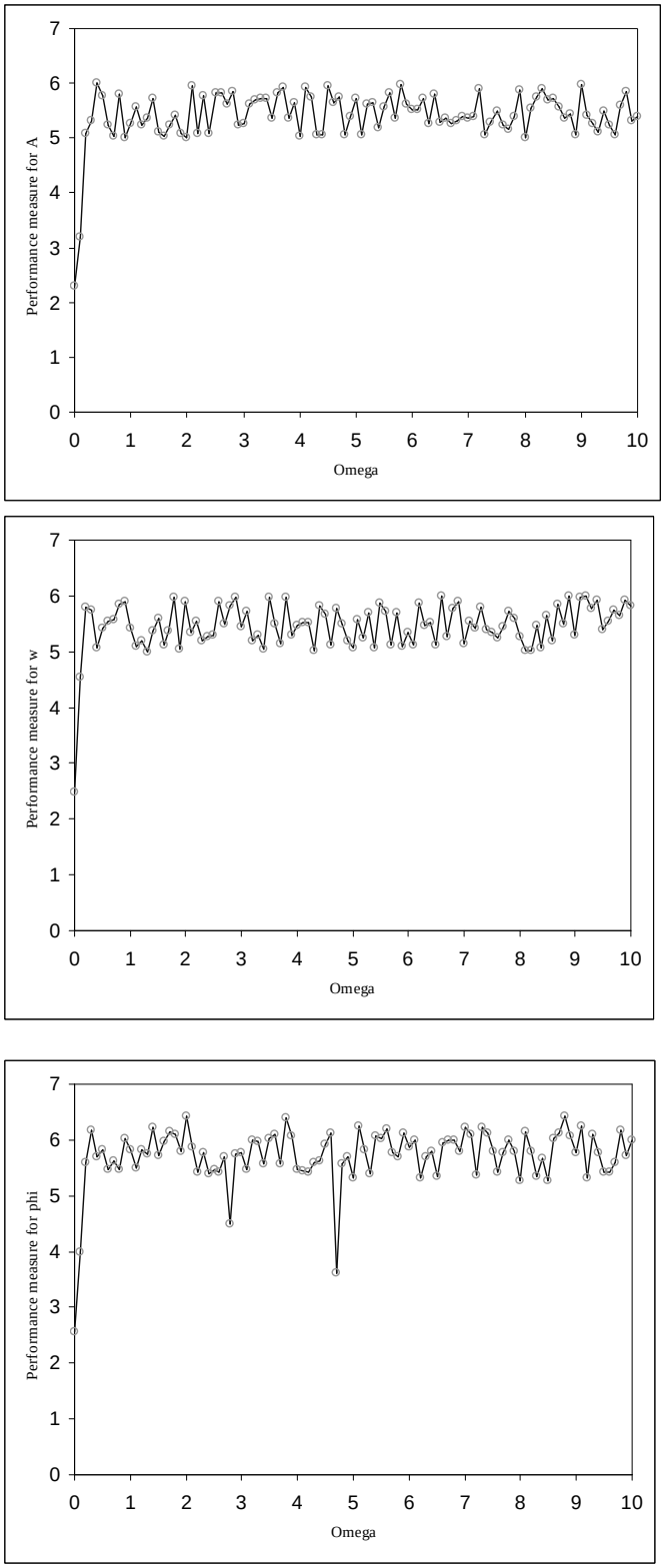


Figure 21: Plots of the performance measures $P(A)$, $P(\omega)$ and $P(\phi)$ against wave velocity.

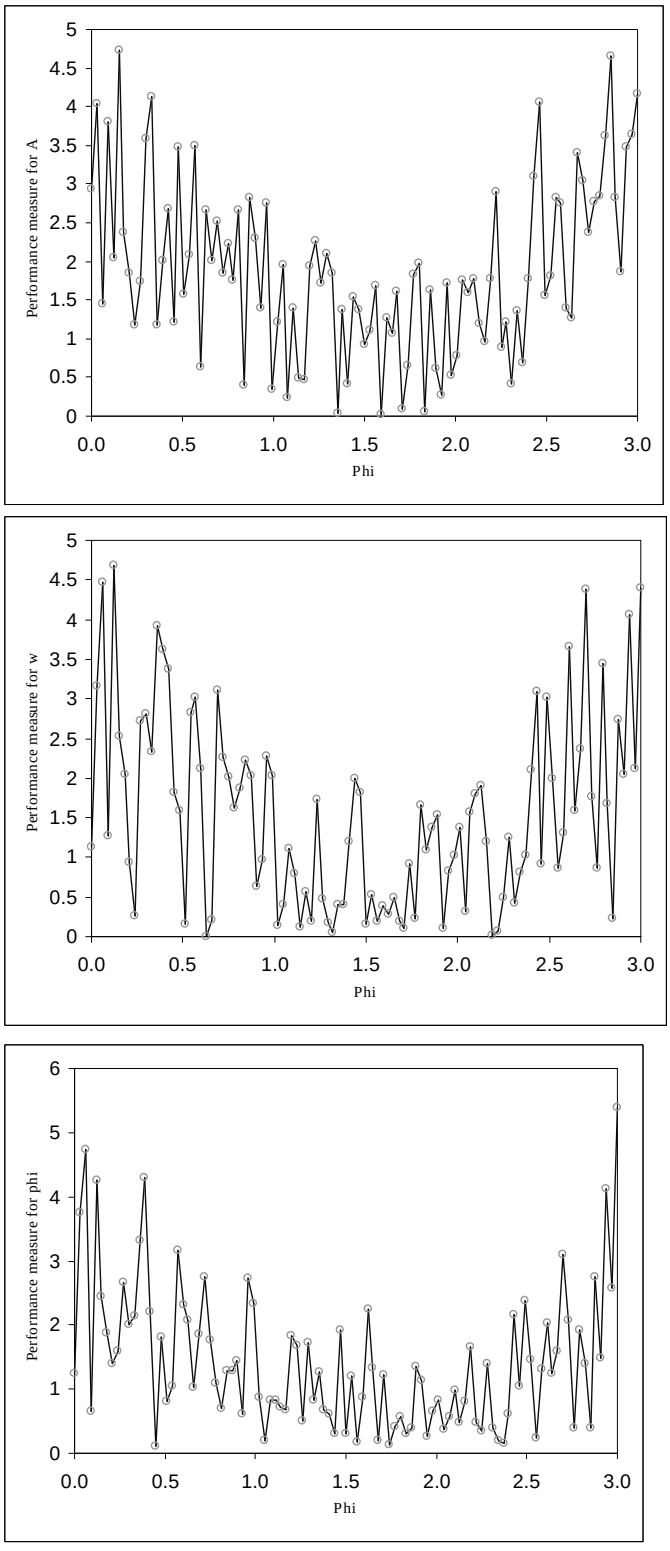
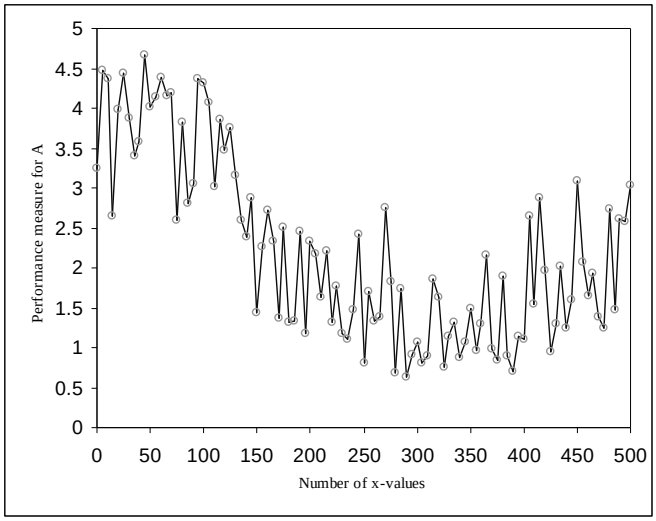
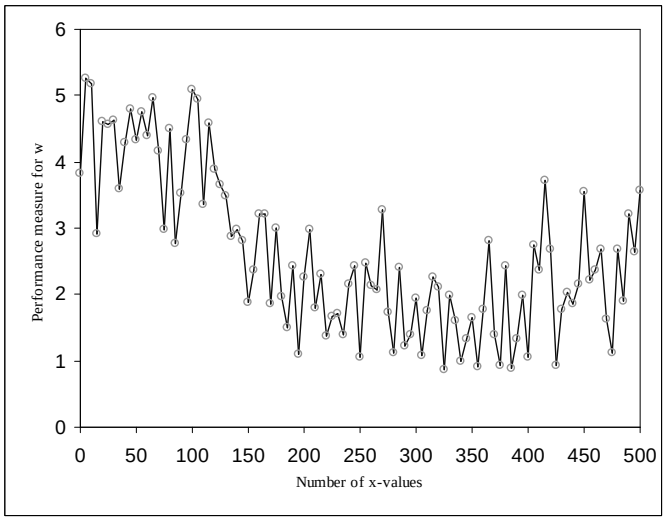


Figure 22: Plots of the performance measures $P(A)$, $P(\omega)$ and $P(\phi)$ against phase angle.





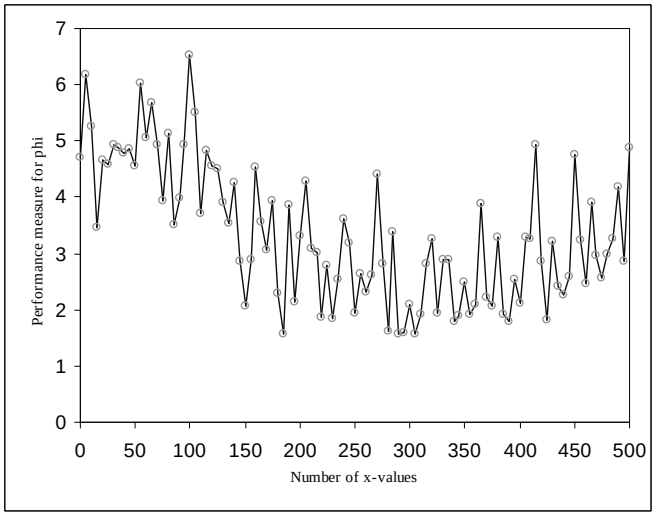


Figure 23: Plots of the performance measures $P(A)$, $P(\omega)$ and $P(\phi)$ against number of data points.

Appendix B: Results for Mathematical and Trigonometric Functions

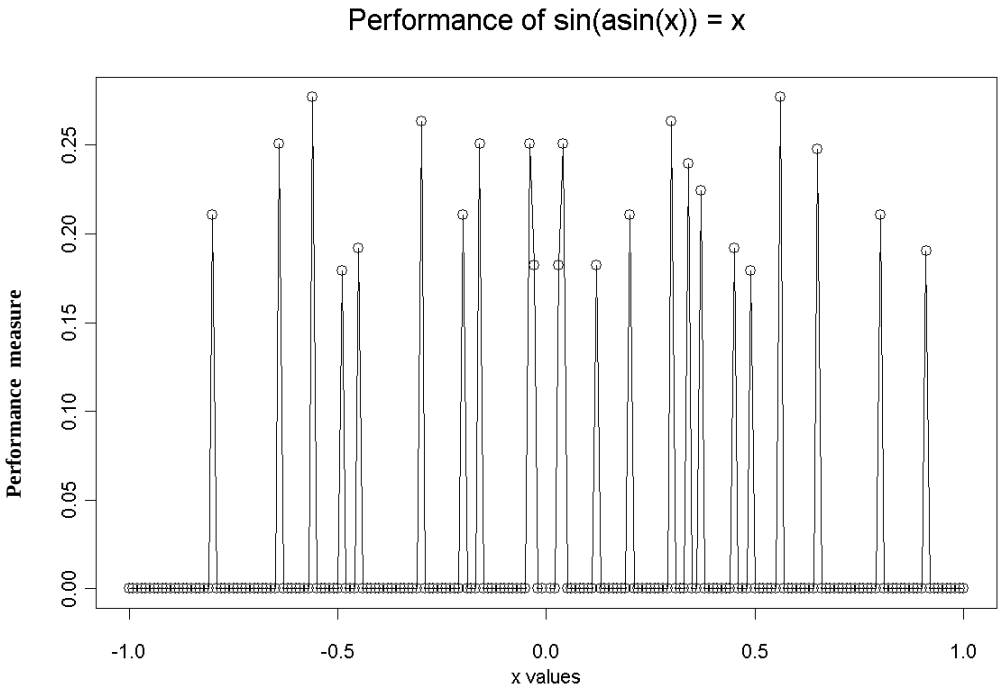


Figure 24: Plot of the performance measure for software test T1.

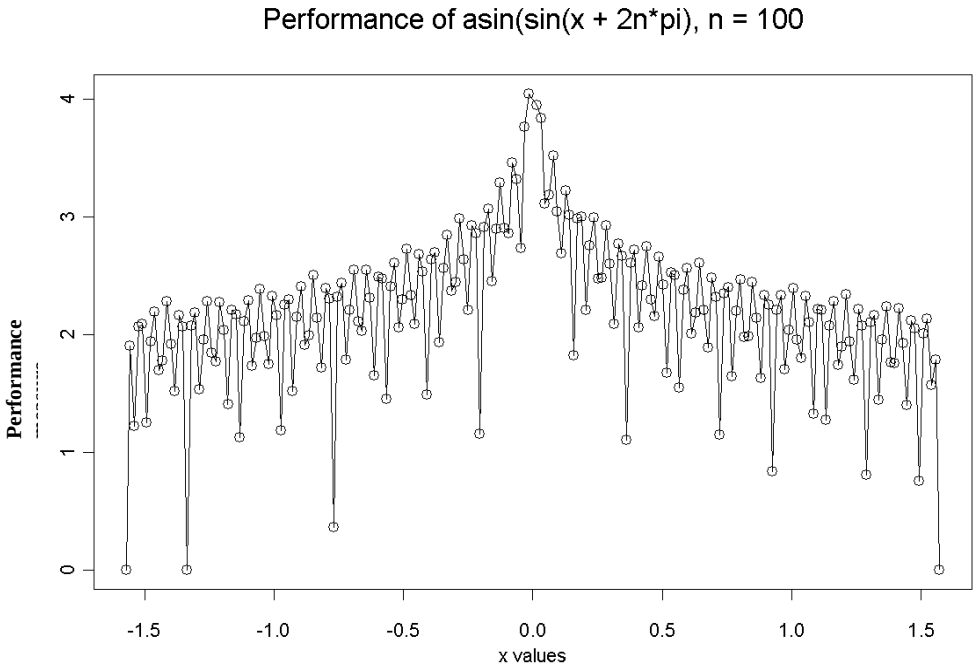


Figure 25: Plot of the performance measure for software test T2.

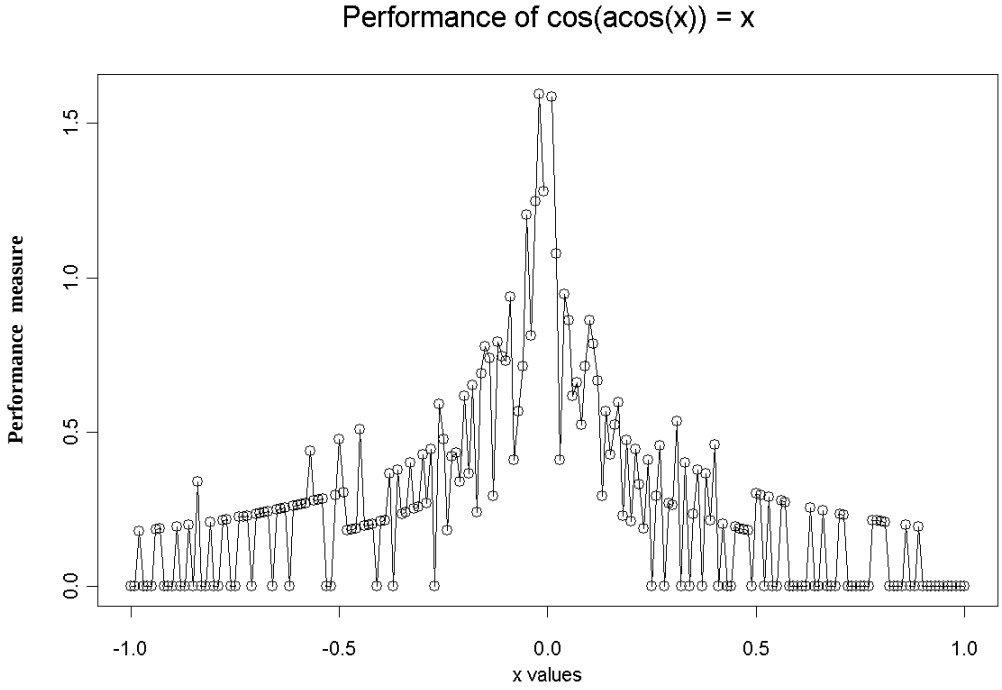


Figure 26: Plot of the performance measure for software test T3.

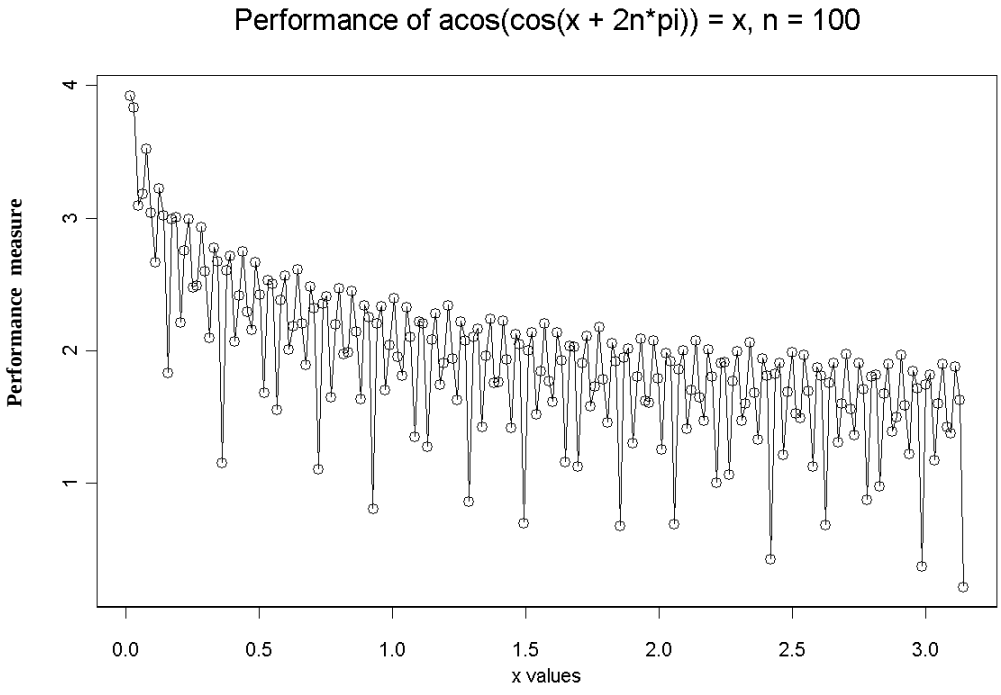


Figure 27: Plot of the performance measure for software test T4.

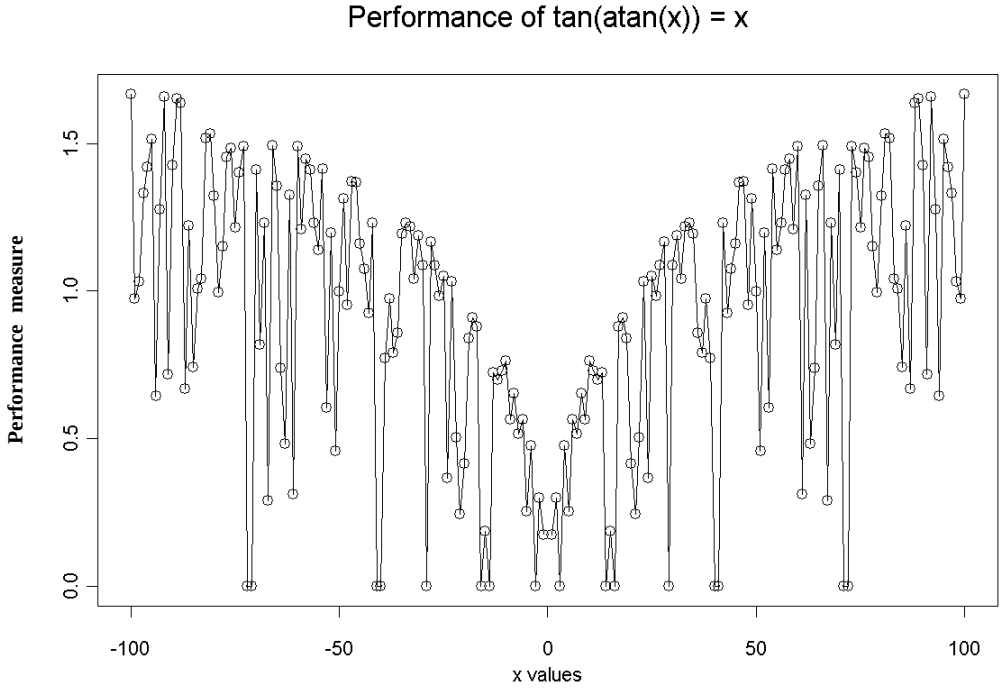


Figure 28: Plot of the performance measure for software test T5.

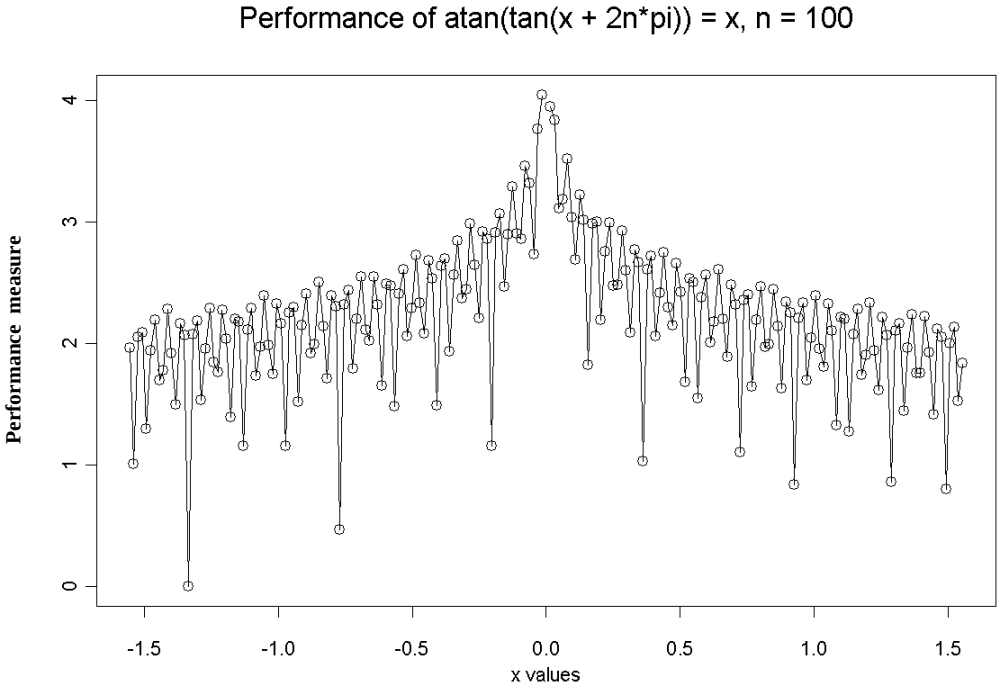


Figure 29: Plot of the performance measure for software test T6.

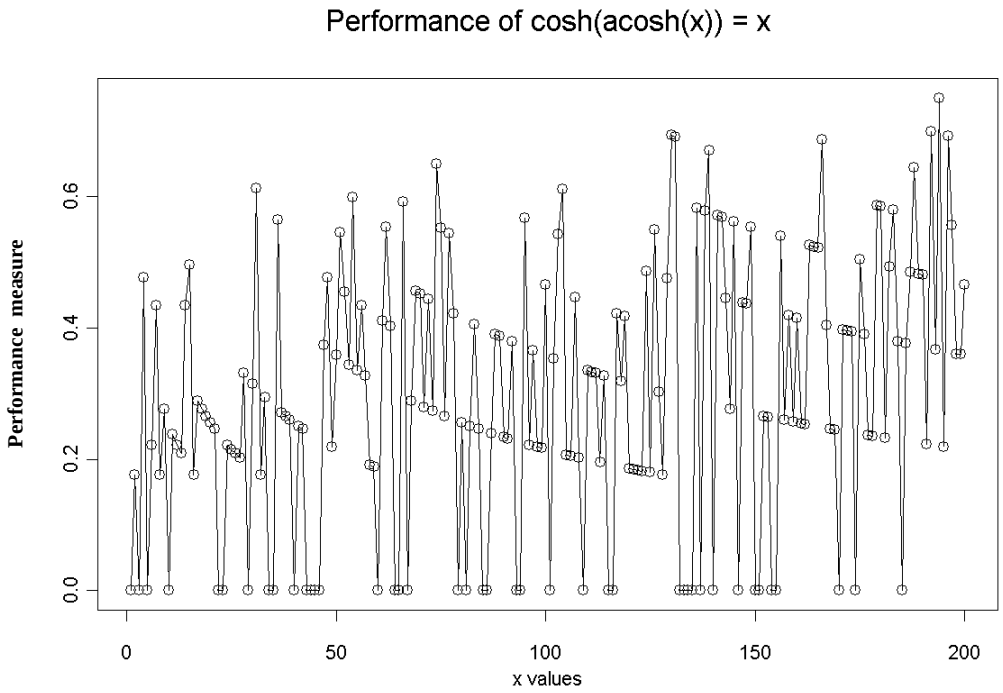


Figure 30: Plot of the performance measure for software test T11.

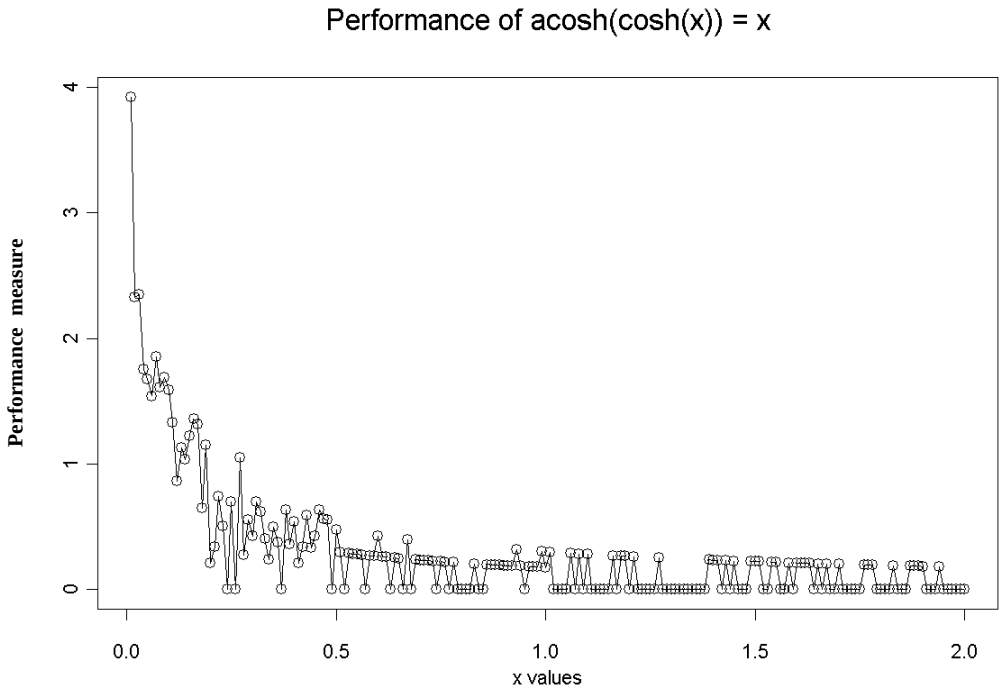


Figure 31: Plot of the performance measure for software test T12.

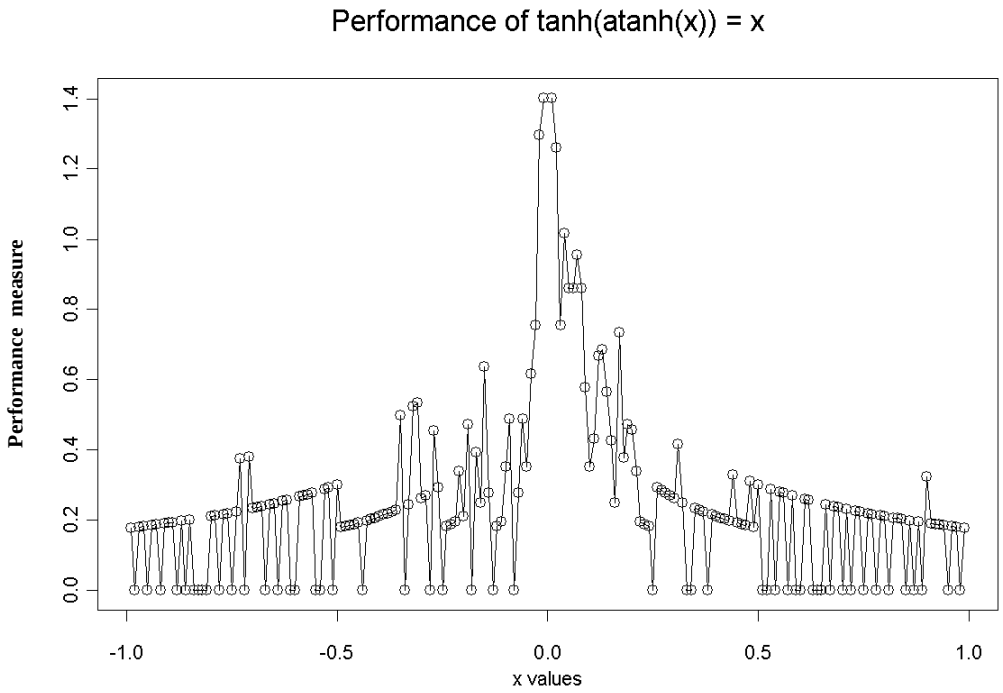


Figure 32: Plot of the performance measure for software test T13.

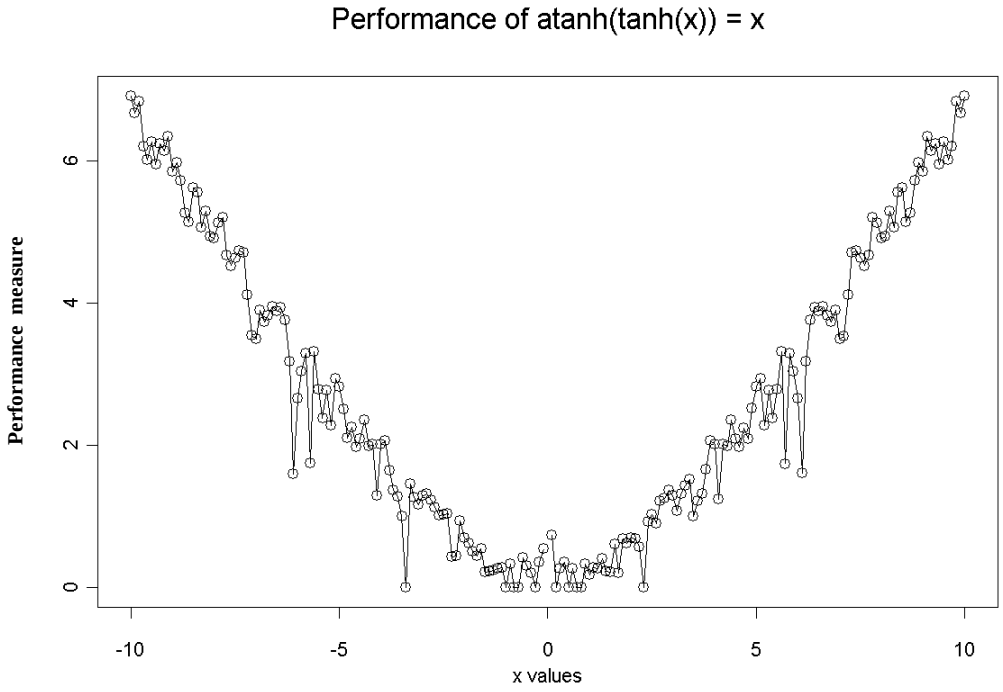


Figure 33: Plot of the performance measure for software test T14.

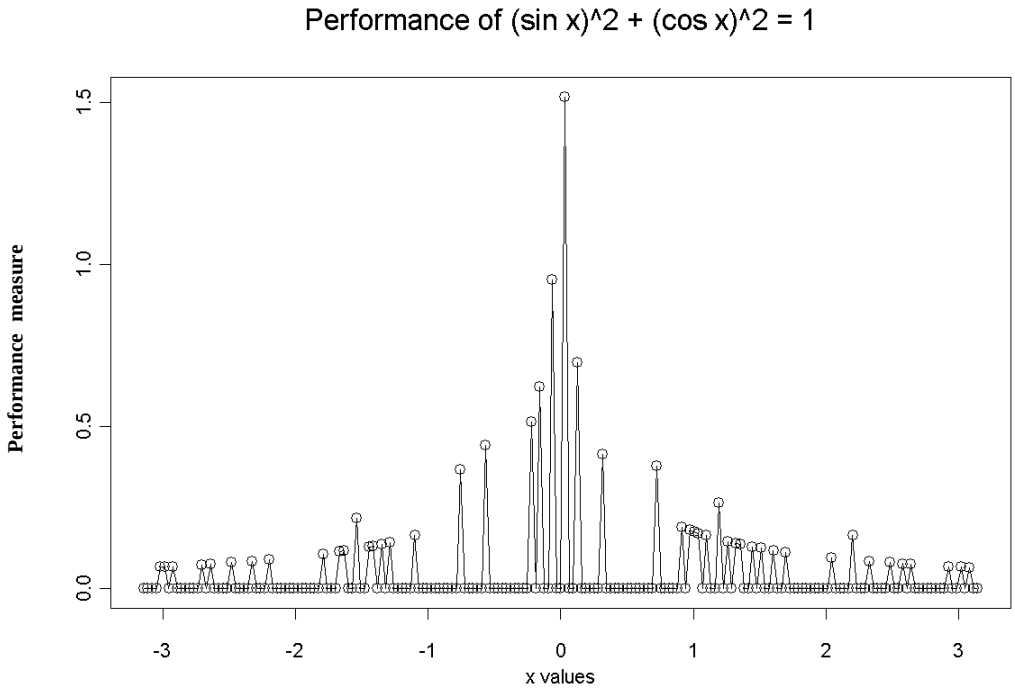


Figure 34: Plot of the performance measure for software test T15.

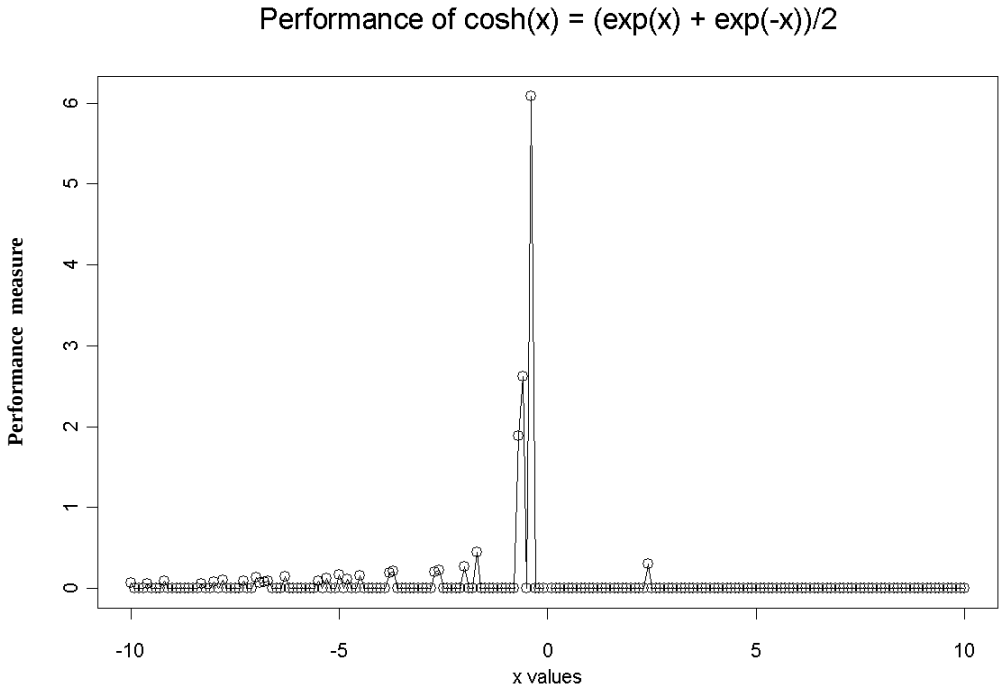


Figure 35: Plot of the performance measure for software test T18.

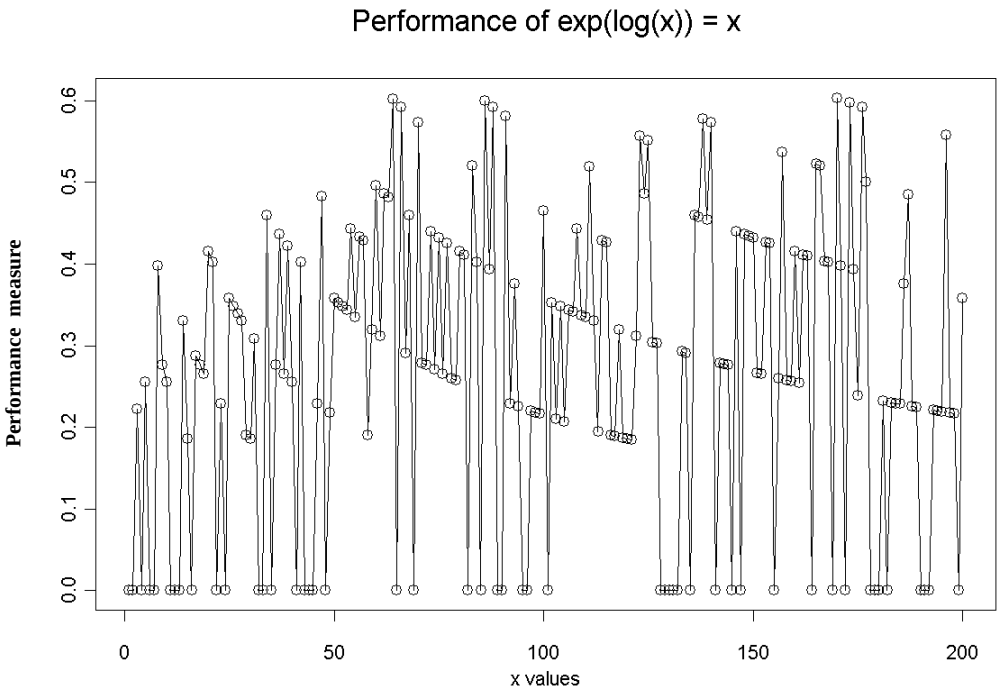


Figure 36: Plot of the performance measure for software test T20.

Appendix C: Results for Statistical Distributions

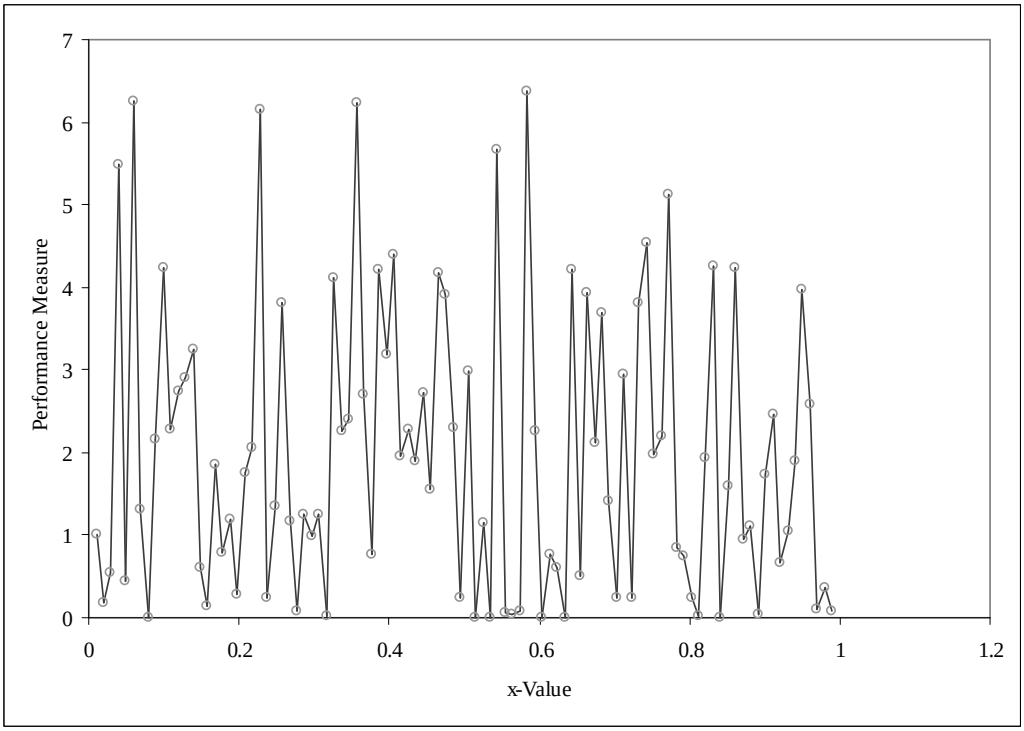


Figure 37: Plot of the performance measure for the comparison of the Beta distribution function with the equivalent IMSL function.

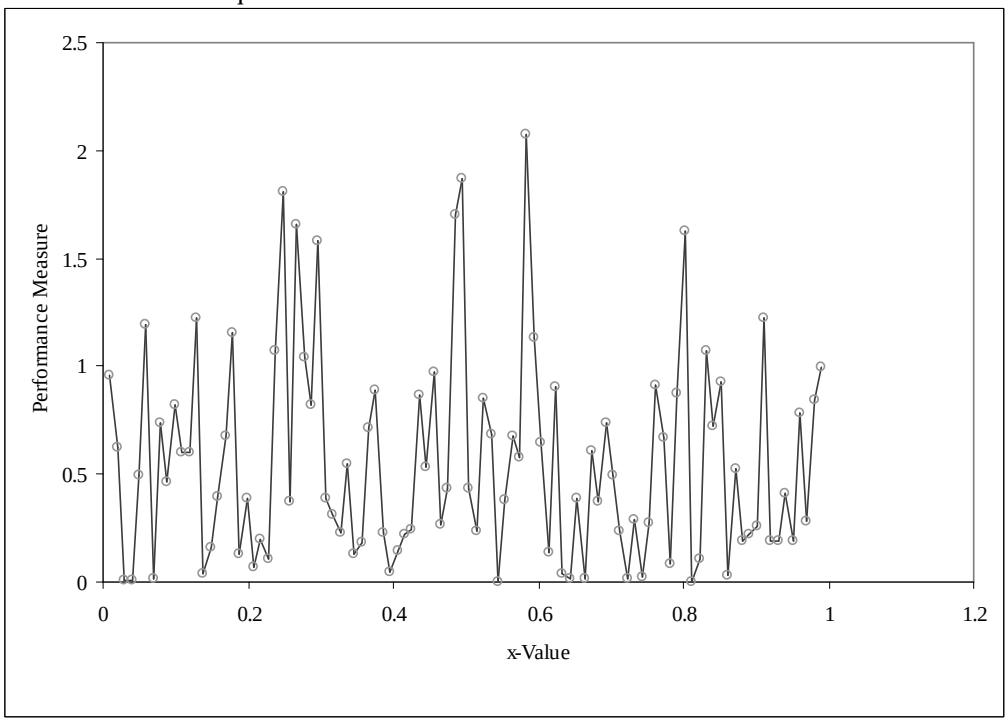


Figure 38: Plot of the performance measure for the comparison of the inverse cumulative Beta distribution function with the equivalent IMSL function.

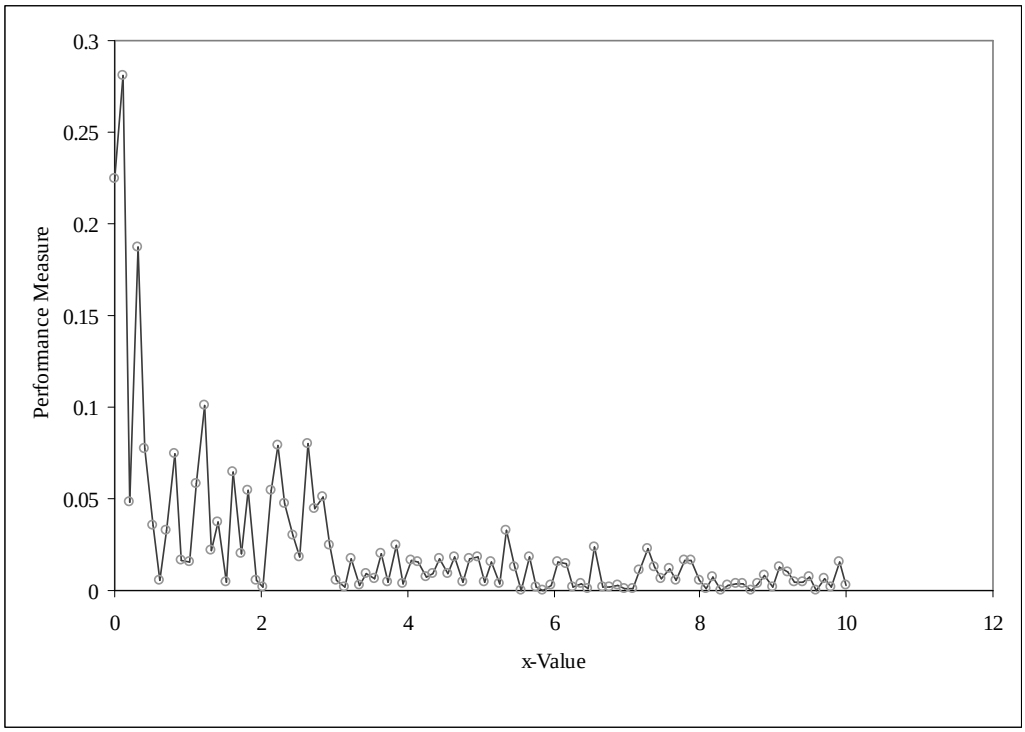


Figure 39: Plot of the performance measure for the comparison of the Chi-squared distribution function with the equivalent IMSL function.

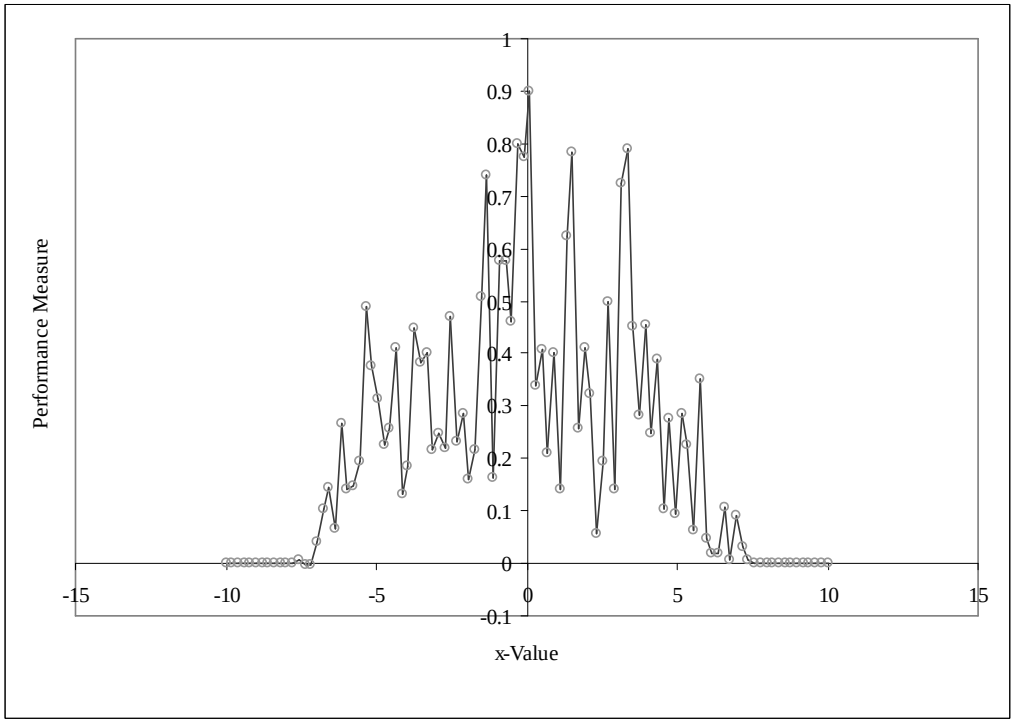


Figure 40: Plot of the performance measure for the comparison of the Gaussian distribution function with the equivalent IMSL function.