NPL Report CMSC 35/04

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

TESTING METHODS OF JAVA LIBRARIES

BY

R E Beckett, M G Cox, M P Dainton,

P M Harris, E G Johnson and G I Parkin

January 2004

Testing Methods of Java Libraries

R E Beckett, M G Cox, M P Dainton, P M Harris, E G Johnson and G I Parkin
Centre for Mathematics and Scientific Computing

January 2004

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 scientific software 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 and libraries that are popular in metrology applications.

We describe the application of a general methodology for testing the numerical correctness of scientific software to specific methods taken from the Java API, the Java LAPACK library (JLAPACK), the Java Numerical Library (JNL) and the JET library. 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 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 *Numerical Software Testing* within the UK Department of Industry's National Measurement System *Software Support for Metrology* Programme 2001–2004.

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 is important to metrology because a poor software implementation can lead to inaccuracy that could have been avoided, and consequently the accuracy of measurement results can be compromised. This work is part of a project performing objective testing of intrinsic and in-built functions included within spreadsheets and other proprietary software packages and libraries that are popular in metrology applications. By reporting the results of these tests, the project aims to contribute to the development of an infrastructure, comprising supporting information and guidelines, to ensure that the use of such software within metrology is made as effective as possible.

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 by users 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 by independent means.

This work is primarily aimed at *users* of software packages and libraries 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 *developers* of the software, for example by highlighting any weaknesses identified in the functions tested.

Java [1, 2] is an object-oriented programming language that has an extensive API (Application Programming Interface) containing standard classes and methods. Java can also call methods from (third party) numerical libraries of which JLAPACK, JNL and JET are examples:

- The JLAPACK project [3] provides the LAPACK [4] numerical subroutines in the form of Java class files, executable by the Java Virtual Machine (JVM), making it possible for Java applications to use established legacy numerical software that was originally written in FORTRAN.
- JNL, a Numerical Library for Java [5], is a set of classes provided by Visual Numerics Inc. for important numerical functions missing from the Java API. The library comprises one *numerical type* class (Complex) and three categories of *numerical function* classes (the special functions class, the linear algebra class and the statistics class).
- The COLT distribution [6] consists of several Java libraries, including
  - the COLT library, providing fundamental general-purpose data structures optimised for numerical data, linear algebra, etc.
  - the JET library, containing mathematical and statistical tools for data analysis, etc.
  - the *RngPack* library, containing random number generators
  - the *VNI* library, containing mathematical functions and complex numbers.

This document records the results of testing performed on a selection of methods taken from the Java API, JLAPACK, JNL and JET libraries. It is a companion document to reports on testing of selected functions from the Microsoft Excel [7], S-PLUS [8] and MathCAD [9] software packages.

The report is organised as follows. In Section 2 the methodology underlying the testing carried out is described. Section 3 contains details of the implementation of the methodology for the testing of Java methods, including specifications of the specific Java methods tested. Section 4 presents, and provides an interpretation of, the results of the testing. Section 5 contains conclusions.

2. Methodology

The procedure employed for evaluating the test software[1] is described in this section. A general methodology [10] for evaluating the numerical correctness of the results produced by scientific software has been used, together with some other complementary tests.

The basis of the general approach is the design and use of *reference data sets* and corresponding *reference results* to undertake "black box" testing of the software. The reference data sets and results are generated in a manner that is 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.

The complementary tests comprise:

- consistency checks: for example, the consistency of a forward calculation followed by its inverse calculation is investigated[2]
- continuity checks: for example, for functions that use different evaluation methods over sub-ranges of the function argument(s), the continuity of the evaluation methods across the sub-range boundaries is investigated
- spot checks against tabulated values
- checks of solution characterisations.

The methodology is described in several reports [10, 11, 12, 13], and is illustrated by a case study [14] and its application to testing selected functions from the Microsoft Excel [7], S-PLUS [8] and MathCAD [9] software packages.

The testing procedure is divided into six stages:

1. documenting the specification of the test software
2. interfacing to the test software
3. specification of reference data sets
4. specification of performance measures and testing requirements
5. generation of reference pairs, i.e., reference data and corresponding reference results
6. presentation and interpretation of performance measures.

These stages adopt existing recommendations [10], with the exception that the first two stages have been modified to suit testing from the perspective of a user of proprietary software packages and libraries. During the software development process, the first two stages would be *specification of the test software*, and *implementation of the test software*, though in practice these are carried out with varying degrees of formality. The application of this procedure by a

[1] The term *test software* used throughout this report refers to the software under test, and not the software employed to do the testing.

[2] An example would be comparing values of $\sin^{-1}\{\sin x\}$ against x.

software developer is presented in a case study [14]. Recording the results of the first two stages is important because it serves to define the basis of the testing undertaken, and to make clear any assumptions about the test software.

The testing process involves the use of software in addition to the test software itself. This is for

- generating reference data sets and corresponding reference results
- evaluating and presenting quality metrics and performance measures
- applying complementary software tests (such as those listed above).

The remainder of this section contains an overview of the implementation of each of the six stages to the testing of a selection of methods taken from the Java API, JLAPACK, JNL and JET libraries. In the description that follows, a knowledge of the Java environment and its terminology is assumed.

2.1 Documenting the specification of the test software

All the Java, JLAPACK, JNL and JET methods implemented in the above libraries have (online) documentation that describes the purpose of the method, together with details of input and output parameters. Often, there is little information about the nature of the numerical algorithms used, and consequently a black-box approach to testing is especially appropriate.

The information provided in the documentation is regarded in this work as providing the basis of a *specification* of the method, against which the method is tested. As an example, the JNL class `VisualNumerics.math.statistics` contains **`standardDeviation`**, a method for the calculation of (sample) standard deviation. The documentation for the method states that the method "returns the sample standard deviation" and includes the following information:

> **Parameters:**
> x - The input double vector for which the sample standard deviation is desired.
>
> **Returns:**
> A scalar of type double containing the sample standard deviation of the input vector x.
>
> **Algorithm:**
> This method returns the sample standard deviation of a set of values. The sample standard deviation is the square root of the sample variance of the set of values.

Furthermore, in the description of the JNL **`variance`** method, it is stated that the sample variance is calculated as **`1/(n-1)*SUM(`**x_i **`- xmean)`**^{2}.

The input and output parameters of the **`standardDeviation`** method are well defined, and in this example a description of the algorithm used is given. In other cases, however, it is a matter of interpretation to derive an *unambiguous* specification from a statement of the purpose of the method.

From the specification it is possible to set down the requirements for generating reference data sets and corresponding reference results for testing the method. In this example, these are to

- generate reference data sets consisting of values for the input vector "x"
- provide corresponding reference results for the output (sample standard deviation) computed from the input in accordance with the specification of the **`standardDeviation`** method described above.

The specification of each method tested is constructed from its online documentation in a similar manner.

2.2 Interfacing to the test software

The implementations tested in this work are methods contained within the classes

- `java.lang.Math` of the Java API [1, 2]
- `org.netlib.lapack.DGELSX` of the JLAPACK library [3]
- `VisualNumerics.math.Statistics` of the JNL library [5]
- `VisualNumerics.math.Sfun` of the JNL library [5]
- `cern.jet.stat.Descriptive` of the JET library [6]
- `cern.jet.stat.Probability` of the JET library [6].

To access a specific method the class from which it is derived must be imported from the appropriate library. For example, to access the JLAPACK method **`dgelsx`** (for computing the minimum-norm solution to a real linear least-squares problem) contained in the class `org.netlib.lapack.DGELSX`, the following statement is included near the top of the Java program:

```
import org.netlib.lapack.DGELSX;
```

In this way the method **`dgelsx`** can be accessed in the same way as methods contained in the Java API.

Reference data sets and corresponding reference results are provided in the form of ASCII text files. A Java program is written to

1. read the reference data sets and corresponding reference results
2. apply the test software (method) to the reference data set to obtain a test result
3. evaluate a performance measure or quality metric to compare the test result with the reference result
4. write the value of the performance measure or quality metric to another ASCII text file.

2.3 Specification of reference data sets

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. In this way the testing undertaken is able to say something about the "fitness for purpose" of the software within the context of its application within such processes. It is, therefore, important that the range of inputs over which a function 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 are generated. Sequences will be *graded* 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 software is based on a poor choice of mathematical algorithm.

For example, consider the problem of finding the least-squares best-fit straight-line 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
- the "magnitude" of the measurement errors used to define the data values y_i, $i = 1, \ldots, m$, described by the standard deviation σ of the probability distribution of which the residual errors are samples.

Each of the parameters *m*, *a*, *b* and σ may be varied in turn, with the remaining parameters taking nominal values, and the performance of a Java method for solving the above least-squares problem investigated as a function of each parameter. Where data sets are found for which the performance of the method 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 will know under which circumstances it is safe or unsafe to use the method.

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

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

can be applied with *m* uniformly-spaced values of *x* in the range $[-\pi/2, +\pi/2]$, with (integral values of) *m* and *n* are regarded as performance parameters for the 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 of reference data sets to which the test software is applied. The evaluation of quality metrics and performance measures, and their presentation as graphs (Section 2.6), is here undertaken using a combination of Java methods and Microsoft Excel graphing (charting) facilities. This enables large numbers of tests to be undertaken automatically and quickly.

By relating the values of the quality 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 [10]:

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

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

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

$$\|\mathbf{y}\| = \sqrt{\sum_{i=1}^{m} y_i^2}\,.$$

2. Provided $\mathbf{y}^{\text{ref}} \neq \mathbf{0}$, the *relative* difference between the test and reference results, i.e.,

$$d_R(\mathbf{x}) = \frac{\left\| \mathbf{y}^{\text{test}} - \mathbf{y}^{\text{ref}} \right\|}{\left\| \mathbf{y}^{\text{ref}} \right\|}. \tag{3}$$

3. A *performance measure* that accounts for factors such as the computational precision and conditioning of the problem. The performance measure

$$P(\mathbf{x}) = \log_{10}\left(1 + \frac{d_R(\mathbf{x})}{\kappa(\mathbf{x})\eta}\right), \tag{4}$$

where $\kappa(\mathbf{x})$ measures the "problem degree of difficulty" defined by the data set $\mathbf{x}$, and η is the computational precision[3] is used [10]. $P(\mathbf{x})$ indicates the number of decimal digits 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 (1),

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

Then,

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

is an *absolute* measure of consistency between the functions f and g, for $x \neq 0$

$$d_R(x) = \frac{|h(x) - x|}{|x|} \tag{6}$$

is a *relative* measure of consistency, and

$$P(x) = \log_{10}\left(1 + \frac{d_R(x)}{\eta}\right) \tag{7}$$

is a *performance measure* for consistency.

It is less straightforward to use the metrics (5), (6) and (7) to make *quantitative* statements about the test software, although *qualitative* judgements are possible. Furthermore, care is necessary: the result that the quality metric $d_A(x)$ given by (5) is zero does not imply that the individual functions f and g (e.g., "sin" and "$\sin^{-1}$") 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.

2.5 Generation of reference pairs

For the testing of methods 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*. In the case of methods solving regression problems, the basis of the data generators is the *null-space method* [10, 13], in which the solution characterisation is used to construct families or classes of data sets possessing the same solution. Data generators have been implemented for this purpose using Java [15], and may be accessed on the web at

[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 that today employ IEEE arithmetic, $\eta = 2^{-52} \approx 2 \times 10^{-16}$, corresponding to approximately 16-digit working.

http://www.npl.co.uk/ssfm/theme2/rds/

In the application of the complementary tests, particularly the consistency checks, reference values are available directly. For example, for the consistency check based on the identity (1), 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 metrics for this test given in Section 2.4.

For many of the intrinsic functions tested, the test results returned by the methods are compared with values obtained from other implementations of the functions. The NAG library [16] has been used for this purpose. Many of the routines contained in the NAG library are accompanied by statements about the (relative) accuracy of the results returned and, consequently, it is appropriate to regard the routines as *reference software* for the testing considered here.

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 those defined by (2), (3) and (4) in 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 plotted against the performance parameter, i.e., as a *performance profile*.

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

1. the range of values of the performance parameter that correspond to the application of interest
2. whether the values of the quality metric over the identified range of the performance parameter 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 **`linearFit`** method (from the JNL library) for ordinary straight-line regression is (a parameter λ defining) the slope of the solution straight line. A graph is provided showing values of the performance measure $P(\mathbf{x})$ given by (4) against this parameter. The graph can be used to

- identify values of the slope for which the performance of the **`linearFit`** method meets a specified accuracy requirement, e.g., the results are to be accurate to a specified number of significant figures
- assess the performance of the **`linearFit`** method for data sets characterised by a particular range of slope values.

For the complementary software tests, quality metrics, such as those defined by (5), (6) and (7) in Section 2.4, are presented (in tabular or graphical form) as a function of the input argument x. For example, for the consistency check based on the identity (1), with $x \in [-\pi/2, +\pi/2]$, a plot is provided of the performance measure $P(x)$ against x. 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. Testing the Java Methods

The particular implementations of the methods tested in this work are those contained within the Java API, and the JLAPACK, JNL and JET libraries. The methods are grouped as follows:

1. the basic sample statistics of sample (arithmetic) mean and sample standard deviation
2. ordinary straight-line regression
3. evaluation of mathematical and trigonometric functions
4. evaluation of statistical distributions.

In the following sections we indicate the particular methods tested, and present the performance parameters, performance measures and complementary tests appropriate to each group of methods.

3.1 Sample (arithmetic) mean and standard deviation

3.1.1 Methods tested

The methods **`average`** and **`standardDeviation`** provided in the JNL library may be used to calculate, respectively, the sample (arithmetic) mean and standard deviation for a set of observations. A specification of these methods is given in Appendix A.1.

The methods **`mean`** and **`sampleVariance`** provided in the JET library may also be used for these calculations. The **`sampleVariance`** method returns the sample variance rather than the sample standard deviation, but the required (test) result is obtained by taking the square root of the result returned by the method. A specification of these methods is given in Appendix A.1.

The JET library contains *two* methods called **`sampleVariance`**. The methods are *overloaded*, possessing different input parameters and for which the (online) documentation suggests that different formulae for the sample variance are implemented. For the first, the sample variance s^2 is evaluated using

$$s^2 = \frac{SS - \frac{S}{m}S}{m-1}, \tag{8}$$

in terms of input values[4]

$$SS = \sum_{i=1}^{m} x_i^2, \qquad S = \sum_{i=1}^{m} x_i \qquad \text{and} \qquad m.$$

For the second, the variance is evaluated using

$$s^2 = \frac{\sum_{i=1}^{m} (x_i - \bar{x})^2}{m-1}, \tag{9}$$

in terms of input values x_i, $i = 1, \ldots, m$, and $\bar{x}$, the (sample) mean of the data.[5] It is known [17] that the above formulae (8) and (9) for s^2, although being mathematically equivalent, possess different numerical properties, with (8) providing a *numerically unstable* formula for calculating the sample variance and (9) a numerically stable formula for this calculation. Consequently, in order to demonstrate the (known) numerical instability associated with

[4] The JET library provides methods for evaluating the sum *S* and sum of squares *SS*.

[5] The sample mean may be calculated using the JET method **`mean`**.

formula (8), we have in this work considered the (first) method implementing this formula, in which the input parameters are *m*, *S* and *SS*.

The JET library also provides a method **`sampleStandardDeviation`** for the sample standard deviation, whose input parameters are the size *m* of the sample and the sample variance s^2. This method [18] returns a value for the quantity

$$\hat{s} = C_m s,$$

where

$$C_m = \frac{\left((m-1)/2\right)^{1/2}\Gamma\left((m-1)/2\right)}{\Gamma\left(m/2\right)}$$

and $\Gamma(m)$ is the gamma function given by

$$\Gamma(m) = \int_0^1 \left(\ln(1/x)\right)^{m-1}\, dx.$$

The method is not tested as part of this work because the formula implemented by the method, as given above, is *mathematically* different from the formula (e.g., (8) or (9)) that is (usually) assumed for the specification of the calculation of the sample standard deviation. It follows that, for a given reference pair (Section 2.5) that is consistent with the specification of the calculation in terms of (8) or (9), the result returned by the method will be *numerically* different from the reference result. Testing of the method would provide primarily information about the *algorithm* used rather than the *implementation* of the algorithm [19].[6]

3.1.2 Performance parameters and performance measures

A reference pair consists of data $\mathbf{x} = \{x_i: i = 1, \ldots, m\}$ together with corresponding values for the sample mean and sample standard deviation for the data.

The performance parameters used to characterise each reference pair are

- the reference value $\bar{x}^{\text{ref}}$ for the sample mean
- the reference value s^{ref} for the sample standard deviation
- the sample size *m*.

Table 1 lists the values assigned to each performance parameter. Three sequences of reference pairs are used to test software for these computations, where each sequence is generated by setting the values for two of the performance parameters equal to default values and varying the value of the remaining parameter according to the information provided in the table.

The performance measure $P(\mathbf{x})$ for the computations of sample mean and sample standard deviation takes, respectively, the forms

$$P_{\bar{x}}(\mathbf{x}) = \log_{10}\left(1 + \frac{1}{\kappa(\bar{x})\eta}\frac{\left|\bar{x}^{\text{test}} - \bar{x}^{\text{ref}}\right|}{\left|\bar{x}^{\text{ref}}\right|}\right), \qquad \kappa(\bar{x}) = \max\left\{\frac{s^{\text{ref}}}{\left|\bar{x}^{\text{ref}}\right|}, 1\right\},$$

and

[6] The motivation for calculating the sample standard deviation in this way is as follows [18]. Although s^2 given by (8) or (9) is an unbiased estimate of the population variance, its square root is not an unbiased estimate of the population standard deviation. The "correction factor" C_m is included to account for this.

$$P_s(\mathbf{x}) = \log_{10}\left(1 + \frac{1}{\kappa(s)\eta}\frac{\left|s^{\text{test}} - s^{\text{ref}}\right|}{\left|s^{\text{ref}}\right|}\right), \qquad \kappa(s) = \max\left\{\frac{\left|\bar{x}^{\text{ref}}\right|}{s^{\text{ref}}}, 1\right\},$$

where $\bar{x}^{\text{test}}$ and s^{test} are the values returned by test software for the two computations applied to the reference data set **x** [10].

Performance parameter	*Values defining sequences of reference data sets*							
Sample (arithmetic) mean $\bar{x}$	**1**	10	10^2	10^3	10^4	10^5	10^6	10^7
Sample standard deviation *s*	**1**	10	10^2	10^3	10^4	10^5	10^6	10^7
Sample size *m*	10	**50**	100	150	200	300	400	500

Table 1: Values for performance parameters for the specification of reference data sets for the computation of sample (arithmetic) mean and standard deviation. Default values for the performance parameters are shown in bold.

3.2 Ordinary straight-line regression

3.2.1 Methods tested

The ordinary straight-line regression problem is to solve

$$\min_{\mathbf{b}} \sum_{i=1}^{m} e_i^2, \qquad e_i = y_i - f(x_i, \mathbf{b}),$$

where $f(x, \mathbf{b})$ describes a straight-line model. The model may be represented, for example, in the form

$$f(x, \mathbf{b}) = b_1 + b_2 x,$$

although other representations may also be used, for example, in terms of a normalised independent variable.

A linear algebra formulation of the ordinary straight-line regression problem is to solve

$$\min_{\mathbf{b}} \|\mathbf{y} - A\mathbf{b}\|^2 \equiv \min_{\mathbf{b}} \sum_{i=1}^{m} \left(y_i - \mathbf{a}_i^{\mathrm{T}}\mathbf{b}\right)^2, \tag{10}$$

in the case that the observation (design) matrix *A* consists of the row vectors

$$\mathbf{a}_i^{\mathrm{T}} = (1, x_i), \qquad i = 1, ..., m, \tag{11}$$

$\mathbf{b} = (b_1, b_2)^{\mathrm{T}}$ and $\mathbf{y} = (y_1, y_2, \ldots, y_m)^{\mathrm{T}}$.

The method **`DGELSX`** provided by the JLAPACK library may be used to obtain a solution to a (generic) linear least-squares problem of the form (10). In this work we consider testing the numerical correctness of the method to solve the (specific) linear least-squares problem in which the matrix *A* takes the form given by (11). A specification of the method, and details of how it is used to solve the ordinary straight-line regression problem, are given in Appendix A.2.1.

The method **`linearFit`** provided by the JNL library may also be used to solve the ordinary straight-line regression problem. A specification of the method, and details of how it is used to undertake this calculation, are given in Appendix A.2.2.

3.2.2 Performance parameters and performance measures

A reference pair consists of data $\mathbf{x} = \{(x_i, y_i): i = 1, \ldots, m\}$ together with values $\mathbf{e}^{\text{ref}} = (e_1^{\text{ref}}, e_2^{\text{ref}}, \ldots, e_m^{\text{ref}})^{\text{T}}$ for the residuals evaluated at the solution. Residuals are chosen as reference values because they describe the solution to the ordinary straight-line regression problem in a way that is independent of the particular parameterisation for the straight-line model used by the test software.

The performance parameters used to characterise each reference pair are

- the coordinate x_c defining the centroid of the x-values $\{x_i: i = 1, \ldots, m\}$ of the data $\mathbf{x}$
- the factor λ defining the slope $\tan \lambda\pi$ of the solution line
- the number m of points in $\mathbf{x}$
- the length L of the (shortest) interval containing the x-values $\{x_i: i = 1, \ldots, m\}$ of the data $\mathbf{x}$
- the measurement standard deviation σ.[7]

Sequences of reference pairs are used to test software for ordinary straight-line regression, where each sequence is generated by setting the values for four of the performance parameters equal to default values and varying the value of the remaining parameter. The values for the performance parameters that define these sequences are listed in Table 2.

The performance measure for the ordinary straight-line regression problem takes the form:

$$P_{\mathbf{e}}(\mathbf{x}) = \log_{10}\left(1 + \frac{1}{\kappa(\mathbf{x})\eta}\frac{\left\|\mathbf{e}^{\text{test}} - \mathbf{e}^{\text{ref}}\right\|}{\left\|\mathbf{e}^{\text{ref}}\right\|}\right), \qquad \kappa(\mathbf{x}) = \max\left\{\frac{\|\mathbf{y}\|}{\left\|\mathbf{e}^{\text{ref}}\right\|}, 1\right\},$$

where $\mathbf{e}^{\text{test}} = (e_1^{\text{test}}, e_2^{\text{test}}, \ldots, e_m^{\text{test}})^{\text{T}}$ are the residuals returned by the test software.

Performance parameter	*Values defining sequences of reference data sets*							
Coordinate x_c defining the centroid of the x-data	**1**	10	10^2	10^3	10^4	10^5	10^6	10^7
Factor λ defining the slope $\tan \lambda\pi$ of the line	–0.33	–0.25	–0.20	–0.10	0.10	0.20	**0.25**	0.33
Number of points m	10	**50**	100	150	200	300	400	500
Length L of the interval containing the data	**1**	50	100	200	400	600	800	1000
Measurement standard deviation σ	**1**	2	3	4	5	6	7	8

Table 2: Values for performance parameters for the specification of reference data sets for the computation ordinary straight-line regression. Default values for the performance parameters are shown in bold.

[7] Specifically, reference data are generated for which the residuals $\mathbf{e}^{\text{ref}}$ are close to numbers sampled randomly from a Gaussian distribution with mean zero and standard deviation σ.

3.3 Evaluation of mathematical functions

3.3.1 Methods tested

Methods are provided in the `java.lang.Math` class of the Java API, and the `VisualNumerics.math.Sfun` class of the JNL library for evaluating common mathematical and trigonometric functions.

The particular methods considered in this work are listed in Appendix A.3, with for each a short statement of its purpose.

3.3.2 Performance parameters and performance measures

The following consistency tests for the mathematical and trigonometric functions are applied:

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

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

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

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

T5. $\tan(\text{atan}(x)) = x, -100 \le x \le 100.$

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

T7. $\text{atan2}(y, x) = \text{atan}(y/x), x > 0, y > 0.$

T8. $\text{atan2}(y, x) = \text{atan}(y/x) + \pi, x < 0, y > 0.$

T9. $\text{atan2}(\sin(x + 2\pi), \cos(x + 2\pi)) = x, -\pi < x \le \pi.$

T10. $\sinh(\text{asinh}(x)) = x, 0 \le x \le 100.$

T11. $\text{asinh}(\sinh(x)) = x, 0 \le x \le 1.$

T12. $\cosh(\text{acosh}(x)) = x, 1 \le x \le 100.$

T13. $\text{acosh}(\cosh(x)) = x, 0 \le x \le 1.$

T14. $\tanh(\text{atanh}(x)) = x, -1 < x < 1.$

T15. $\text{atanh}(\tanh(x)) = x, -10 \le x \le 10.$

T16. $\sin^2(x) + \cos^2(x) = 1, -\pi \le x \le \pi.$

T17. $\tan(x) = \sin(x)/\cos(x), -10 \le x \le 10.$

T18. $\sinh(x) = (\exp(x) - \exp(-x))/2, -1 \le x \le 1.$

T19. $\cosh(x) = (\exp(x) + \exp(-x))/2, -1 \le x \le 1.$

T20. $\tanh(x) = \sinh(x)/\cosh(x), -1 \le x \le 1.$

T21. $\exp(\ln(x)) = x, 0 \le x \le 200.$

T22. $\ln(\exp(x)) = x, -1 \le x \le 1.$

T23. $\log(x) = \ln(x)/\ln(10), 0 \le x \le 200.$

In each case, the performance measure $P(x)$ given by (7) in Section 2.4,

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

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 $d_R(x)$ is significantly greater than η)
2. by identifying discontinuities in value and/or slope of the performance measure as a function of its argument x
3. by noting trends in the values of the performance measure, e.g., periodic behaviour of the performance measure.

3.4 Evaluation of statistical distributions

3.4.1 Methods tested

Methods are provided in the `VisualNumerics.math.Statistics` class of the JNL library, and the `cern.jet.stat.Probability` class of the JET library for evaluating cumulative and inverse cumulative functions for a number of common probability distributions.

The cumulative functions are used to calculate the (lower-tailed) probability that a random variable following a given probability distribution lies within a specified range, i.e., to evaluate

$$p = \int_{-\infty}^{x} f(u)\, du, \tag{12}$$

where x is given, and f is the probability density function for the random variable. The probability p is calculated by applying a numerical (quadrature) scheme, using series (and other) approximations and recurrence relations and combinations of these. The inverse cumulative functions are used to calculate x to satisfy (12), where p and the probability density function f are given.

The particular methods considered in this work for evaluating cumulative probability distributions and their inverses are listed in Appendix A.4, with for each a short statement of its purpose.

3.4.2 Performance parameters and performance measures

Comparisons between test and reference software for the following statistical distributions are undertaken:[8]

- Beta distribution with parameters $a = b = 2$ and variate x with $0 < x < 1$
- Binomial distribution with parameters $k = 100$ and $n = 150$ and probability x with $0 < x < 1$
- χ^2 distribution with ten degrees of freedom and variate x with $0 < x < 10$
- F distribution with five and ten degrees of freedom and variate x with $0 < x < 10$
- Gamma distribution with parameters $a = 1$ and $b = 2$ and variate x with $0 < x < 1$
- Normal (Gaussian) distribution with mean zero and variance unity and variate x with $-2 < x < 2$
- Poisson distribution with $k = 10$ terms and mean x with $0 < x < 10$
- Students'-t distribution with five degrees of freedom and variate x with $-10 < x < 10$

In each case, a reference pair consists of the value x for the variate of the distribution together with (a reference value for) the lower-tailed probability p for the distribution corresponding to x. The reference value is provided by a NAG routine [16]. Moreover, the departure between the

[8] Here, "x" is used as a generic symbol rather than, for example, the "natural" symbol p for probability.

test and reference results (p^{test} and p^{ref}, respectively) is measured by an absolute difference[9] as in (2) given in Section 2.4, i.e.,

$$d_A(x) = \left| p^{\text{test}} - p^{\text{ref}} \right|.$$

The following consistency tests for the statistical distributions are also applied (see Appendix A.4). In each case, the value of x (with $0 < x < 1$) is compared with that of $h(x)$ (see Section 2.4) where $h(x)$ is assigned to be:[10]

T24. `Probability.normal(0,1,Probability.normalInverse(x))`

T25. `Probability.normalInverse(Probability.normal(a,b,x))`

T26. `Statistics.FCdf(Statistics.inverseFCdf(x,1,2),1,2)`

T27. `Statistics.inverseFCdf(Statistics.FCdf(x,1,2),1,2)`

In each case, the performance measure $P(x)$ given by (7) is used to make judgements about the test software in the ways described in Section 4.2.

4. Presentation and Interpretation of Results

4.1 Sample (arithmetic) mean and standard deviation

Figures 1–3 in Appendix B[11] show values of the performance measure for test software for the calculation of sample (arithmetic) mean for, respectively, the performance parameters reference sample mean, reference sample standard deviation, and sample size (Table 1).[12] For all reference data sets, and both functions tested, the performance measure takes values between zero and one. This indicates that the numerical accuracy of the results returned by the test software is comparable to that expected from optimally stable (reference) software for this calculation and these data sets.

Figures 4–6 in Appendix C show the results for test software for the calculation of sample standard deviation and the same performance parameters. For some of the reference data sets the performance value for both functions tested takes values between zero and one (Figures 5 and 6).

However, for the sequence of reference data sets characterised by the performance parameter reference sample mean (Figure 4), the performance measure for the JET function takes values that exceed unity (for some of the data sets in this sequence) and, furthermore, exhibits a tendency to *increase* with the performance parameter. This behaviour of the performance measure indicates that the JET function implements a *numerically unstable* formula for the calculation of sample variance (from which the sample standard deviation is evaluated) [15,

[9] An *absolute* difference is chosen (rather than a measure based on a relative difference) because the result of the calculation is a probability that always lies in the range zero to one, and that is appropriately calculated to a constant absolute accuracy.

[10] See Section 4.4 regarding comments on attempts to implement consistency checks involving methods for the Student's-*t* distribution function and its inverse.

[11] In the figures in this appendix, the results for the JNL function are "hidden behind" those for the JET function.

[12] In the figures in the appendices, the values of the performance measures are joined by straight lines. The lines have no meaning and are included only for visualisation purposes. Furthermore, it should be noted that for many of the graphs the scale of the "independent" variable (performance parameter) is non-uniform. Consequently, care is required in the interpretation of the straight-line segments.

17].[13] For the same sequence of reference data sets the performance measure for the JNL function takes values between zero and one and, consequently, it is likely that the function implements a numerically stable formula.

The second JET **`sampleVariance`** method described in Section 3.1.1 implements the numerically stable formula (9) for the sample variance. Apart from the issue of numerical correctness (which is the concern of this work), there would appear to be no obvious reason (such as "ease" of use) for a user to choose one method over the other. Consequently, we would recommend that the second JET **`sampleVariance`** method (implementing formula (9)) is used in preference to the first method considered here (or, alternatively, the JNL method is used).

4.2 Ordinary straight-line regression

Figures 7–11 of Appendix D show values for the performance measure for test software for ordinary straight-line regression for the performance parameters listed in Table 2. Generally (Figures 8–11), the performance measure takes values close to one, indicating that the test software is behaving (qualitatively) essentially like reference software for this computation and these data sets. A value for the performance measure of one is to be interpreted (quantitatively) as meaning that the test software is losing one significant figure of accuracy (in the test results) in addition to the number lost by reference software for the computation.

However, the results illustrated in Figure 7 include values for the performance measure for both functions tested that (a) exceed zero (appreciably), and (b) exhibit a tendency to increase with the performance parameter (the coordinate of the centroid of the data x-values). The behaviour of the performance parameter for each function tested is to increase in proportion to the value of the performance parameter, the constant of proportionality being one for the JLAPACK function and two for the JNL function.

The issues for test software for this computation that affect the numerical correctness of the results returned by such software, when the results are the *residual deviations* associated with the solution, are (a) model parametrisation, and (b) the numerical algorithm used to solve the least-squares problem (10) [15]. The behaviour of the performance measure for the JLAPACK function is likely to be a consequence of the model parametrisation, which is in terms of "natural" unnormalised variables, whereas that for the JNL function is likely to be the result of a combination of the two effects, i.e., the application of a deficient algorithm to a poorly parametrised problem.

For the purpose of evaluating the solution straight-line model (or the residual deviations associated with the solution) the performance of the two test functions is improved by applying them to the reference data following a *transformation* (centering and scaling) of the data [15]. This will make the JLAPACK function behave like reference software for evaluating the model, but will not remove any deficiency in the algorithm used in the JNL function to solve the least-squares problem.[14]

[13] This is confirmed by the documentation for the function (Section 3.1.1), which states that that the sample variance is calculated using the formula (8). This formula, although mathematically correct, is a numerically unstable way for undertaking the calculation.

[14] The issue is that the functions tested return the *coefficients* (slope and intercept) that define the solution, and the residual deviations are evaluated in terms of these coefficients. The testing reported here highlights both numerical aspects of the implementations of the functions as well as aspects of how the functions are applied to obtain the required test results (residual deviations).

4.3 Mathematical functions

Figures 12–22 in Appendix E show values for the performance measure for a selection of the complementary tests (T1–T23) listed in Section 3.3.2. Table 3 gives the maximum value of the performance measure observed for each test.

For many of the tests (perhaps T13 and T15 are the only exceptions) the values of the performance measure are likely to be sufficiently small for them not to be of concern for most (metrology) applications. However, the graphs do highlight some "interesting" behaviour, as follows:

- For complementary test T3 (which involves odd functions f for which $f(-x) = -f(x)$), the values of the performance measure are not symmetric about the $x = 0$ axis (Figure 12).[15] For other tests, including T5 (Figure 13), symmetry does appear to be present.
- For several of the complementary tests, including T11 (Figure 15), the performance measure exhibits "systematic behaviour" (in the sense that subsets of the values follow distinct "trends"), although interspersed with (occasional) zero values.
- For several of the complementary tests, including T23 (Figure 22), the performance measure shows discontinuities or "jumps" between the trends followed by different subsets of the values.[16]

The systematic behaviour observed might be explained by the fact that an innate sensitivity associated with the complementary checks is not being fully accounted for in the performance measure. For some input values x the floating-point numbers involved are such that the complementary test is exact; for other values the problem sensitivity will indicate an inflation of the computational precision η that manifests itself as a non-zero value for the performance measure. 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.

Finally, Figure 23 in Appendix E shows values for the performance measure for complementary test T16 obtained when that test is executed on the same machine but using different versions of the Java run-time environment, viz., versions 1.4.0, 1.3.1 and 1.2.2. For all three versions and all values of the variable x considered in the test, the values of the performance measure are between zero and one. This indicates that the differences between the test result, viz., "$\sin^2(x) + \cos^2(x)$", and the reference result, viz., "1", are all of the order of the computational precision η, and so *qualitatively*, the numerical behaviour is the same when the test is executed for the three versions. However, the results illustrated in the figure indicate that, *quantitatively*, the numerical behaviour is different.

This test (involving different versions of the Java run-time environment) is important for an environment, such as Java, for which the times between the release dates for different versions are short compared to that for other established environments and languages, such as FORTRAN. However, no attempt is made to attribute the changes in the results to a particular cause, which might include the implementations of the intrinsic functions "sin" and "cos", the operations used to evaluate the performance measure, or the way the results are written to file.

[15] This is particularly evident for the ranges $-1 < x < -0.5$ and $0.5 < x < 1$.

[16] For complementary test T23 the discontinuities appear to arise at $x = 10$ and $x = 100$, corresponding to powers of the base (10) for the mathematical function (log) considered in this test.

T1	*T2*	*T3*	*T4*	*T5*	*T6*	*T7*	*T8*	*T9*	*T10*	*T11*	*T12*
0.4	2.1	1.5	2.6	2.0	2.1	0.0	0.4	1.9	0.8	0.5	0.8
T13	*T14*	*T15*	*T16*	*T17*	*T18*	*T19*	*T20*	*T21*	*T22*	*T23*	
4.0	0.5	7.2	0.5	0.5	1.9	0.5	0.6	0.7	2.0	0.5	

Table 3: Maximum value of the performance measure for each complementary test (T1–T23) listed in Section 3.3.2.

4.4 Statistical distributions

Figures 24–31 in Appendix F show values for the performance measure for a selection of the comparisons and complementary tests listed in Section 3.4.2. As for the mathematical and trigonometric functions (Section 4.3), the graphs show systematic trends interspersed with occasional zero values and discontinuities in the values of the performance measure. Unlike the mathematical and trigonometric functions, however, the requirements on the accuracy of the results returned by the functions for statistical distributions are likely to be much less severe.[17] Consequently, it is expected that the values of the performance measure reported here are likely to be sufficiently small for most (metrology) applications.

In addition to the complementary tests listed in Section 3.4.2, it was intended to investigate the consistency of the functions **`studentT`** and **`studentTInverse`** (see Appendix A.4) for, respectively, evaluating the cumulative Students'-*t* distribution and its inverse, in the way this was done for the normal and *F* distributions. Our attempt to do this was not successful because we could not see a way of implementing the test: the **`studentT`** function requires the "degrees of freedom" to be specified and the **`studentTInverse`** function the "size of the data set", but the relationship between the two quantities is not made clear in the documentation for the two functions. [18]

5. Conclusions

In this report we have described the application of a general methodology [10] for testing the numerical accuracy of scientific software to specific methods taken from the Java API and related Java numerical packages. Each stage of the methodology, from documenting a specification for each method 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 (black-box) nature of the testing.

It has been the intention in this work not to consider *all* the intrinsic methods included within the numerical libraries identified, 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 the basic descriptive statistic of sample mean and standard deviation, a simple regression problem, 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 method must be

[17] For example, "coverage factors", used in the evaluation of coverage intervals as part of the preparation of statements of measurement uncertainty, that correspond to values of inverse cumulative probability functions (particularly for the Students'-*t* distribution) are typically stated to two (or at most three) significant figures.

[18] Our attempts to establish the relationship, at least empirically, by trial and error were not successful.

interpreted in the context of that method only, and not in the context of other methods of the Java API or the numerical libraries considered. Furthermore, the tests described here have been carried out in such a way that the methods have been used without taking account of information elsewhere, e.g., as contained in publications on Java or information posted on the World Wide Web; only the documentation available on-line as part of the normal Java "environment" was used. This approach is deliberate, since we believe it accords with that adopted by most users generally and within metrology in particular.

The results are intended primarily to help users understand whether for a particular application the methods used are fit for purpose, and to understand the limitations (if any) of those methods. Where the testing has indicated ways in which a user may make better use of a given method to overcome any such limitation, this information has been provided.

Some particular conclusions are as follows:

- The methods tested for the calculation of sample (arithmetic) mean return test results for the reference data sets considered having a numerical accuracy that is *comparable* to that expected from reference software.
- For the calculation of sample standard deviation, a *critical* performance parameter is the (reference) value for the sample mean of the data set. Reference data sets with different values for this performance parameter are used to expose (possible) deficiencies in the formulae used for this calculation. The results of the testing indicate that the JNL library provides a method for this calculation that implements a numerically stable formula. However, it is recommended that the (second) JET method for this calculation, implementing formula (9), is used in preference to the JET method implementing formula (8), the latter having been shown to be numerically unstable.
- For the calculation of ordinary straight-line regression, a *critical* performance parameter is the coordinate of the centroid of the data *x*-values. Reference data sets with different values for this performance parameter can be used to expose (possible) deficiencies in the model parametrisations and numerical algorithms used by methods to undertake this calculation. It is strongly recommended that the data be centred and scaled before undertaking ordinary straight-line regression. It is also recommended that the JLAPACK method should be used in preference to the JNL method for this calculation, as there is evidence from the results of the testing that the latter uses a (numerically) deficient algorithm.
- The application of (so-called) complementary tests to mathematical and statistical functions highlights features of the numerical behaviour of methods for these calculations, including systematic behaviour and discontinuities between regions of systematic behaviour, which may be significant for some applications. In particular, further work is required to investigate the results obtained from the complementary tests T13 and T15, and to resolve the difficulties experienced in using the **`studentT`** and **`studentTInverse`** methods from the JET library.

6. Acknowledgements

This report constitutes one of the deliverables of Project 2.1 of the 2001–2004 NMS *Software Support for Metrology* Programme, and has been funded by the National Measurement System Directorate of the UK Department of Trade and Industry.

We thank Mike Stevens for his help in the preparation of this report.

7. References

[1] Java™ 2 SDK, Standard Edition, Documentation. Copyright © 1995–2003, Sun Microsystems, Inc. (http://java.sun.com/docs/).

[2] B. Joy, G. Steele, J. Gosling and G. Bracha. Java™ Language Specification, Second Edition. Addison-Wesley Publishing Co., 2000.

[3] D.M. Doolin, J. Dongarra and K. Seymour. JLAPACK – Compiling LAPACK FORTRAN to Java. *Scientific Programming* **7** (2), pp 111 – 138, 1999.

[4] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenny, S. Ostouchov and D. Dorensen. *LAPACK User's Guide, Second Edition*. SIAM, Philadelphia, PA, 1995.

[5] JNL, a Numerical Library for Java. Copyright © 1997 - Visual Numerics, Inc.

[6] cern.colt* and cern.jet* packages. Copyright © 1999 CERN - European Organisation for Nuclear Research (http://hoschek.home.cern.ch/hoschek/colt/).

[7] H.R. Cook, M.G. Cox, M.P. Dainton and P.M. Harris. Testing spreadsheets and other packages used in metrology: Testing the intrinsic functions of Excel. *NPL Report CISE 27/99*, September 1999.

[8] J. Barrett and M.P. Dainton. Testing spreadsheets and other packages used in metrology: Testing the intrinsic functions of S-PLUS. *NPL Report CISE 06/00*, September 2000.

[9] J. Barrett and M.P. Dainton. Testing spreadsheets and other packages used in metrology: Testing the intrinsic functions of MathCAD. *NPL Report CISE 05/00*, September 2000.

[10] H.R. Cook, M.G. Cox, M.P. Dainton and P.M. Harris. A methodology for testing spreadsheets and other packages used in metrology. *NPL Report CISE 25/99*, September 1999.

[11] 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.

[12] 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.

[13] 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.

[14] H.R. Cook, M.G. Cox, M.P. Dainton and P.M. Harris. Testing spreadsheets and other packages used in metrology: A case study. *NPL Report CISE 26/99*, September 1999.

[15] P.M. Harris, E.G. Johnson, P.D. Kenward and G.I. Parkin. Testing the numerical correctness of software. *NPL Report CMSC 34/04*, January 2004.

[16] The NAG Fortran Library (Mark 20). Copyright © 2001, Numerical Algorithms Group Ltd.

[17] M.G. Cox, M.P. Dainton and P.M. Harris. Testing functions for the calculation of standard deviation. *NPL Report CMSC 07/00*, 2000.

[18] R.R. Sokal and F.J. Rohlf. *Biometry: the principles and practice of statistics in biological research (third edition)*. W.H. Freeman & Co., New York, 1995.

[19] R.M. Barker, M.G. Cox, P.M. Harris and I.M. Smith. Testing algorithms in standards and METROS. *NPL Report CMSC 18/03*, March 2003.

Appendix A: Specification of Methods

A.1 Sample (arithmetic) mean and standard deviation

The following methods are provided in the `VisualNumerics.math.Statistics` class of the JNL library:

`double average(double x[])` returns the average (mean) for the input vector **`x[]`**.

`double standardDeviation(double x[])` returns the sample standard deviation for the input vector **`x[]`**.

The following methods are provided in the `cern.jet.stat.Descriptive` class of the JET library:[19]

`double mean(DoubleArrayList data)` returns the arithmetic mean of a data sequence:

`Sum( data[i] )/data.size(),`

where **`data.size()`** is the number of elements in the data sequence.

`double sampleVariance(int size, double sum, double sumOfSquares)` returns the sample variance of a data sequence:

`(sumOfSquares – mean*sum)/(size – 1),`

where **`size`** is the number of elements in the data sequence, **`sum`** the sum of the elements, **`sumOfSquares`** the sum of squares of the elements, and **`mean = sum/size`**.

The following methods for sample standard deviation and variance are also provided in the `cern.jet.stat.Descriptive` class of the JET library:

`double sampleVariance(DoubleArrayList data, double mean)` returns the sample variance of a data sequence:

`Sum((data[i]–mean)^2)/(data.size()–1),`

where **`mean`** is the mean of the elements of the data sequence, and **`data.size()`**the number of elements.

`double sampleStandardDeviation(int size, double sampleVariance)` returns the sample standard deviation calculated from the number of elements and the sample variance [18].

A.2 Ordinary straight-line regression

A.2.1 Method **`DGELSX`**

The following method is provided in the `org.netlib.lapack.DGELSX` class of the JLAPACK library:

```
void DGELSX(int m, int n, int nrhs, double a[][],
            double b[][], int jpvt[], double rcond,
            intW rank, double work[], intW info)
```

computes the minimum-norm solution to a (real) linear least-squares problem $AX = B$ using a complete orthogonal factorisation of A.

[19] **`DoubleArrayList`** is a data type provided with the Colt distribution of Java libraries [5].

Parameters:

M (integer input). The number of rows of the matrix *A* (**M** ≥ 0).

N (integer input). The number of columns of the matrix *A* (**N** ≥ 0).

NRHS (integer input). The number of right hand sides, i.e., the number of columns of matrices *B* and *X* (**NRHS** ≥ 0).

A (double precision input/output array, dimension (**LDA**, **N**)). On entry, the **M**-by-**N** matrix *A*. On exit, **A** will be overwritten by details of its complete orthogonal factorisation.

LDA (integer input). The leading dimension of the array *A* (**LDA** ≥ max(1, **M**)).

B (double precision input/output array, dimension (**LDB**, **NRHS**)). On entry, the **M**-by-**NRHS** right hand side matrix *B*. On exit, the **N**-by-**NRHS** solution matrix *X*.

LDB (integer input). The leading dimension of the array *B* (**LDB** ≥ max(1, **M**, **N**)).

JPVT (integer input/output array, dimension **N**). Contains pivoting information.

RCOND (double precision input). Used to determine the effective rank of *A*.

RANK (integer output). The effective rank of *A*.

WORK (double precision workspace array, dimension max(min(**M**, **N**) + 3×**N**, 2×min(**M**, **N**) + **NRHS**)).

INFO (integer output) If zero, indicates successful exit. If –*i*, the *i*th argument has an illegal value.

Prior to calling the method to solve the ordinary straight-line regression problem defined by data **x[]** and **y[]**, the input/output arrays **a[][]** and **b[][]** are assigned as follows:

```
for (j = 0; j < m; j++)
   {
         a[j][0] = 1.0;
         a[j][1] = x[j];
         b[j][0] = y[j];
   }
```

Following the call to the method, the test residuals **e[]** that define the solution line, and are used as the basis of the comparison with the reference results (Section 3.2.2), are evaluated from the result **b[][]** returned by the method as follows:

```
for (j = 0; j < m; j++)
   {
         e[j] = y[j] - (b[0][0] + b[1][0]*x[j]);
   }
```

A.2.3 Method **linearFit**

The following method is provided in the `VisualNumerics.math.Statistics` class of the JNL library:

double[] linearFit(double x[], double y[])

returns the best least-squares fit of a line to data.

Parameters:

x An input double vector containing the *x*-values of the data.

y An input double vector containing the *y*-values of the data.

Returns:

An array of two doubles, call it **`coef[]`**. The equation of the line is then **`y = coef[0] + coef[1]*x`**.

Following the call to the method, the test residuals **`e[]`** that define the solution line, and are used as the basis of the comparison with the reference results (Section 3.2.2), are evaluated from the result **`coef[]`** returned by the method as follows:

```
for (j = 0; j < m; j++)
   {
         e[j] = y[j] - (coef[0] + coef[1]*x[j]);
   }
```

A.3 Mathematical functions

The following methods are provided in the `java.lang.Math` class of the Java API:

`double cos(double x)` returns the cosine.

`double sin(double x)` returns the sine.

`double tan(double x)` returns the tangent.

`double acos(double x)` returns the arc cosine in the range of 0 through π.

`double asin(double x)` returns the arc sine in the range of –π/2 through π/2.

`double atan(double x)` returns the arc tangent in the range of –π/2 through π/2.

`double atan2(double y, double x)` converts rectangular coordinates (*x*, *y*) to polar (*r*, θ). The method computes the phase θ by computing an arc tangent of *y*/*x* in the range of –π to π.

`double exp(double x)` returns the exponential number *e* raised to a power..

`double log(double x)` returns the natural logarithm (base *e* logarithm).

The following methods are provided in the `VisualNumerics.math.Sfun` class of the JNL library:

`double cosh(double x)` returns the hyperbolic cosine.

`double sinh(double x)` returns the hyperbolic sine.

`double tanh(double x)` returns the hyperbolic tangent.

`double acosh(double x)` returns the inverse hyperbolic cosine.

`double asinh(double x)` returns the inverse hyperbolic sine.

`double atanh(double x)` returns the inverse hyperbolic tangent.

`double log10(double x)` returns the base 10 logarithm.

A.4 Statistical distributions

The following methods are provided in the `VisualNumerics.math.Statistics` class of the JNL library:

```
double FCdf(double x,
            double degreesFreedomNumerator,
            double degreesFreedomDenominator)
```

returns the value of the cumulative *F* distribution with specified degrees of freedom.

```
double inverseFCdf(double p,
```

double degreesFreedomNumerator,
double degreesFreedomDenominator)

returns the inverse of the cumulative *F* distribution with specified degrees of freedom.

The following methods are provided in the `cern.jet.stat.Probability` class of the JET library:

double beta(double a, double b, double x)
returns the area from 0 to *x* under the beta probability density function with parameters *a* and *b*.

double binomial(int k, int n, double p)
returns the sum of the terms 0 through *k* of the binomial probability density function with parameters *n* (number of trials) and *p* (probability of success).

double chiSquare(double v, double x)
returns the integral from 0 to *x* of the χ^2 probability density function with *v* degrees of freedom.

double gamma (double a, double b, double x)
returns the integral from 0 to *x* of the gamma probability density function with parameters *a* and *b*.[20]

double normal(double mean, double variance, double x)
returns the integral from $-\infty$ to *x* of the normal (Gaussian) probability density function with specified mean and variance.

double normalInverse(double y)
returns the value *x* for which the integral $-\infty$ to *x* of the normal (Gaussian) probability density function (with mean zero and variance one) is equal to the *y*.

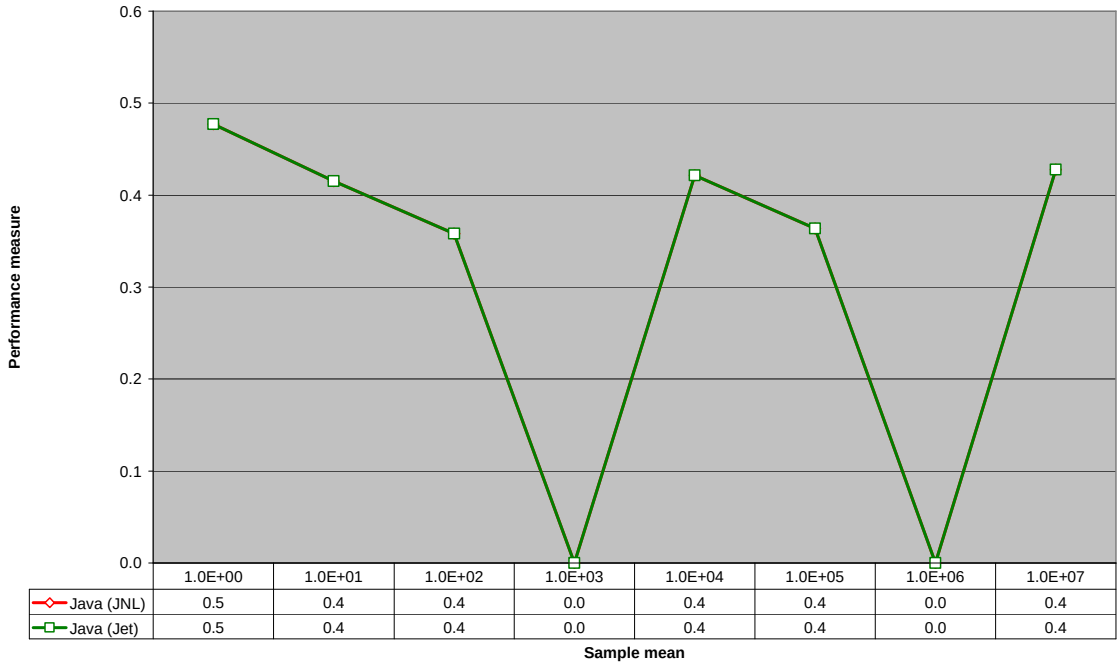
double poisson(int k, double mean)
returns the sum of the first *k* terms of the Poisson distribution with specified mean.

double studentT(double k, double t)
returns the integral from $-\infty$ to *t* of the Students'-*t* distribution with *k* degrees of freedom.

double studentTInverse(double alpha, int size)
returns the value *t* for which the integral from $-\infty$ to *t* of the Students'-*t* probability density function is equal to 1 – alpha/2, where "size is the size of the data set".

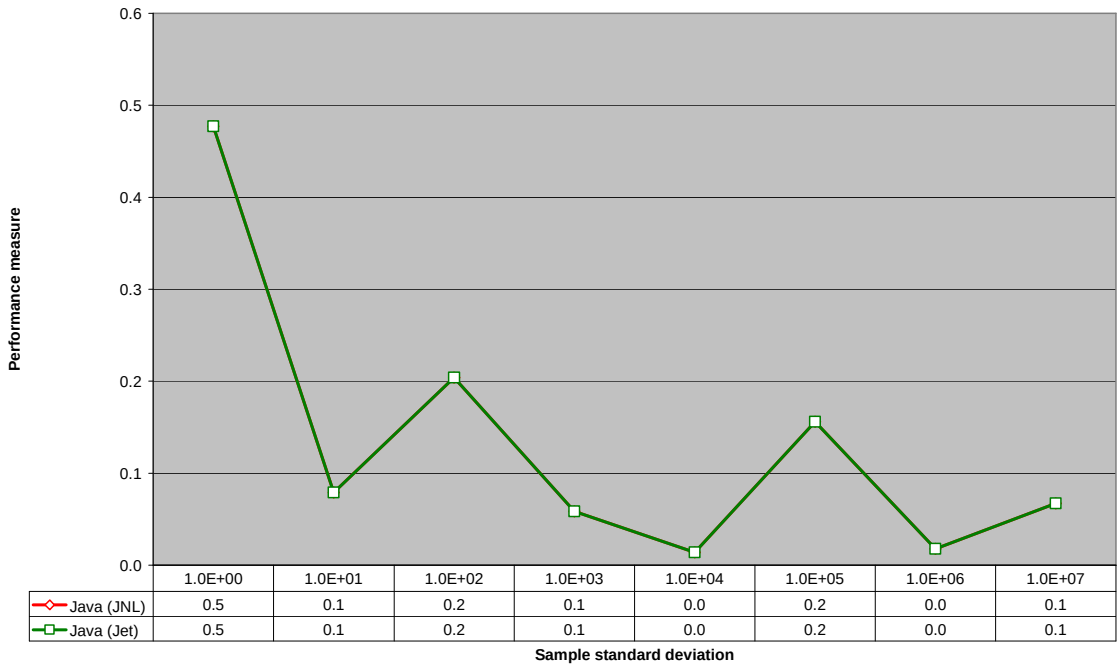
[20] It is important to examine the documentation for this method in order to interpret correctly the parameters *a* and *b*. The equivalent NAG (FORTRAN) routine is defined in terms of parameters α and β, related to *a* and *b* by $\alpha = b$ and $\beta = 1/a$. If the parameters (*a*, *b*) and (α, β) are *assumed* to be identical, we would find appreciable differences when comparing the two software functions.

Appendix B: Results for Sample Mean



	1.0E+00	1.0E+01	1.0E+02	1.0E+03	1.0E+04	1.0E+05	1.0E+06	1.0E+07
Java (JNL)	0.5	0.4	0.4	0.0	0.4	0.4	0.0	0.4
Java (Jet)	0.5	0.4	0.4	0.0	0.4	0.4	0.0	0.4

Figure 1: Values of the performance measure for test software for the computation of the sample (arithmetic) mean. The performance parameter is the reference value for the sample mean.



	1.0E+00	1.0E+01	1.0E+02	1.0E+03	1.0E+04	1.0E+05	1.0E+06	1.0E+07
Java (JNL)	0.5	0.1	0.2	0.1	0.0	0.2	0.0	0.1
Java (Jet)	0.5	0.1	0.2	0.1	0.0	0.2	0.0	0.1

Figure 2: As Figure 1 except that the performance parameter is the reference value for the sample standard deviation.

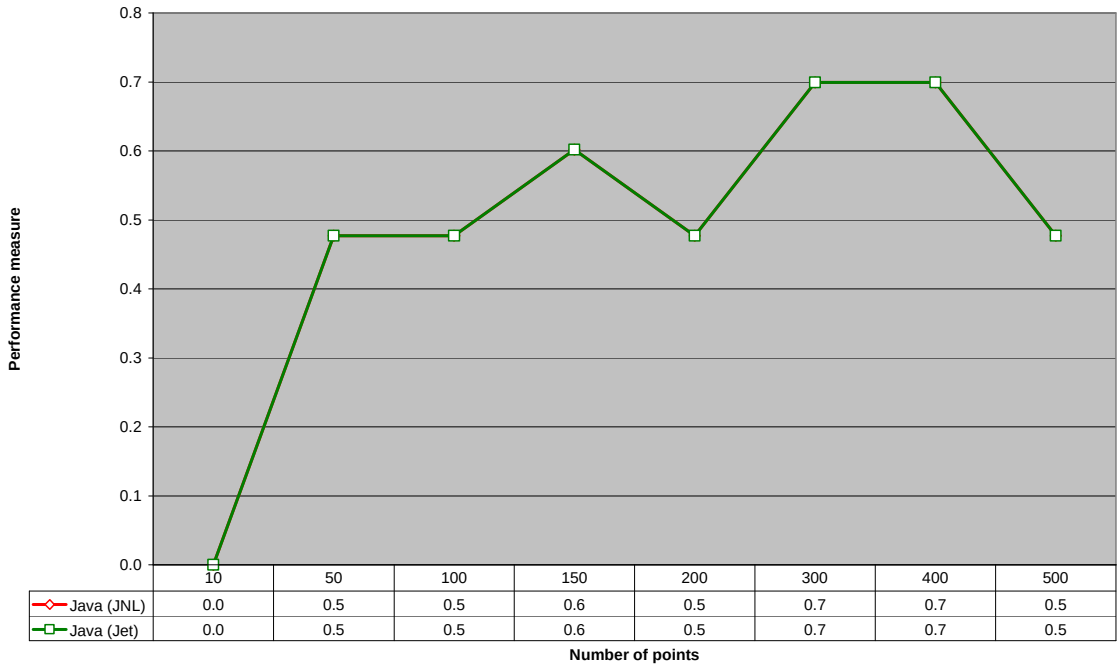
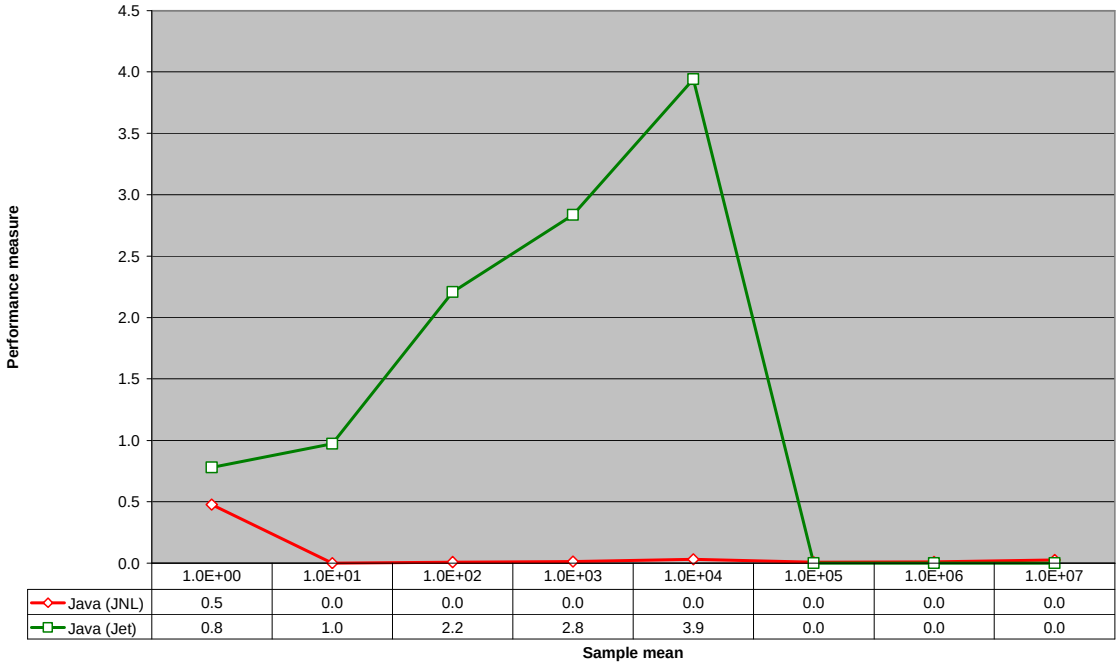


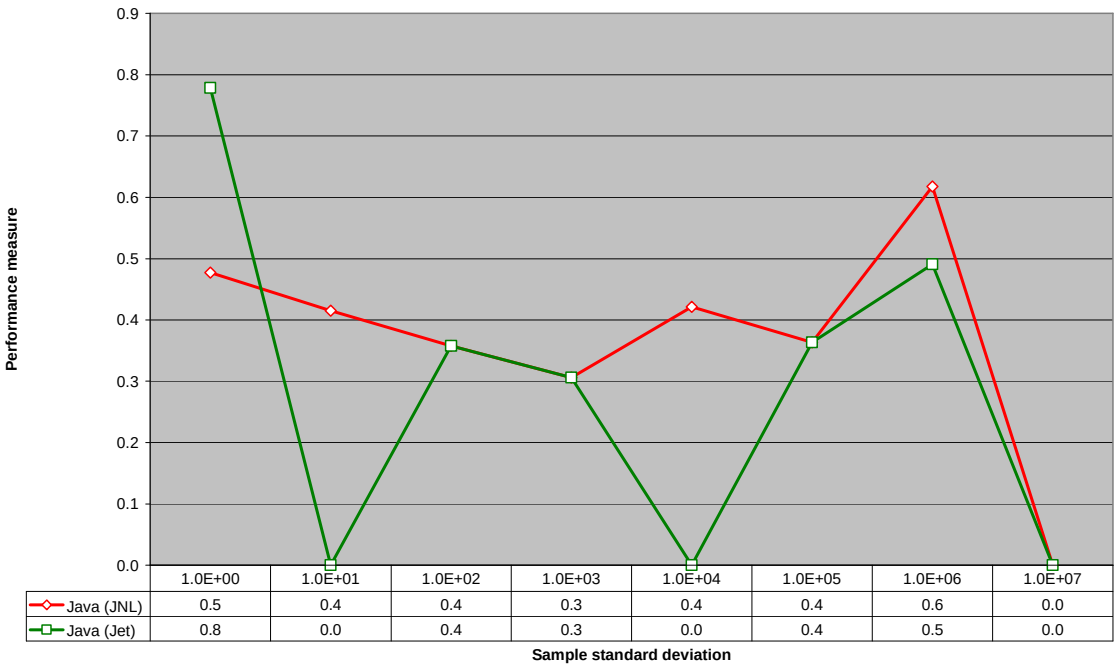
Figure 3: As Figure 1 except that the performance parameter is the sample size.

Appendix C: Results for Sample Standard Deviation



	1.0E+00	1.0E+01	1.0E+02	1.0E+03	1.0E+04	1.0E+05	1.0E+06	1.0E+07
Java (JNL)	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Java (Jet)	0.8	1.0	2.2	2.8	3.9	0.0	0.0	0.0

Figure 4: Values of the performance measure for test software for the computation of the sample standard deviation. The performance parameter is the reference value for the sample mean.



	1.0E+00	1.0E+01	1.0E+02	1.0E+03	1.0E+04	1.0E+05	1.0E+06	1.0E+07
Java (JNL)	0.5	0.4	0.4	0.3	0.4	0.4	0.6	0.0
Java (Jet)	0.8	0.0	0.4	0.3	0.0	0.4	0.5	0.0

Figure 5: As Figure 4 except that the performance parameter is the reference value for the sample standard deviation.

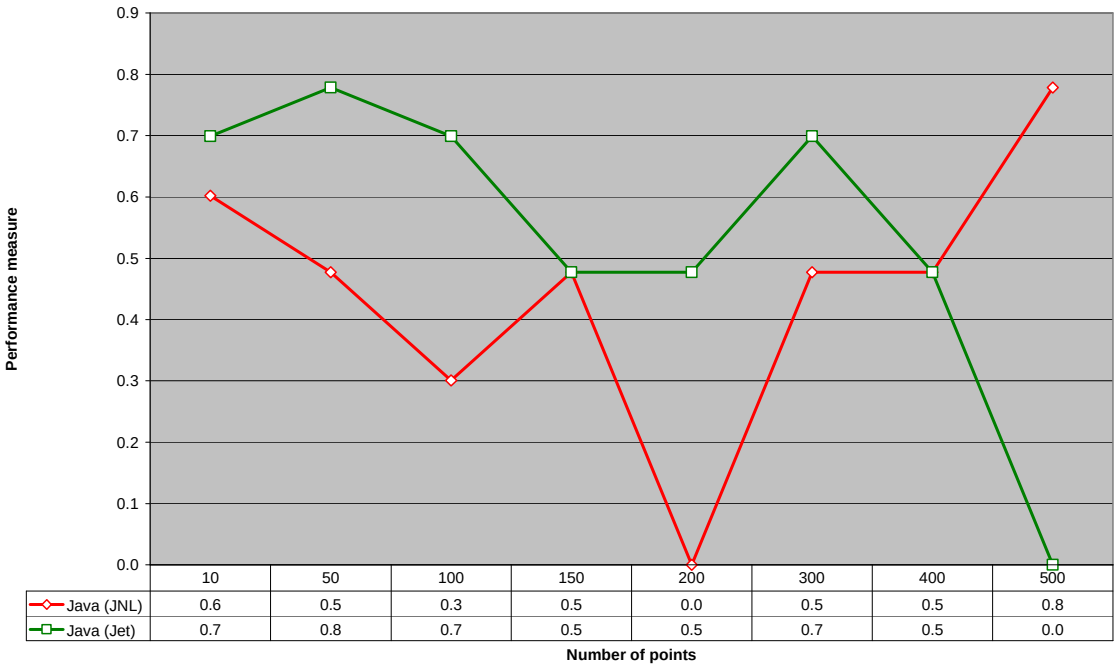
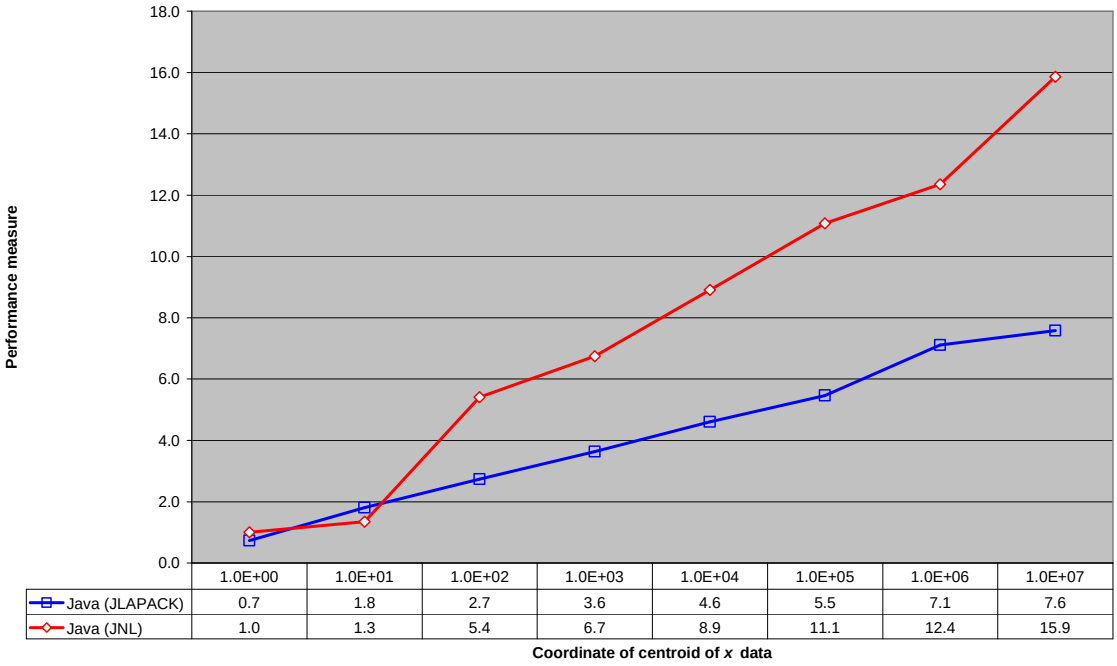


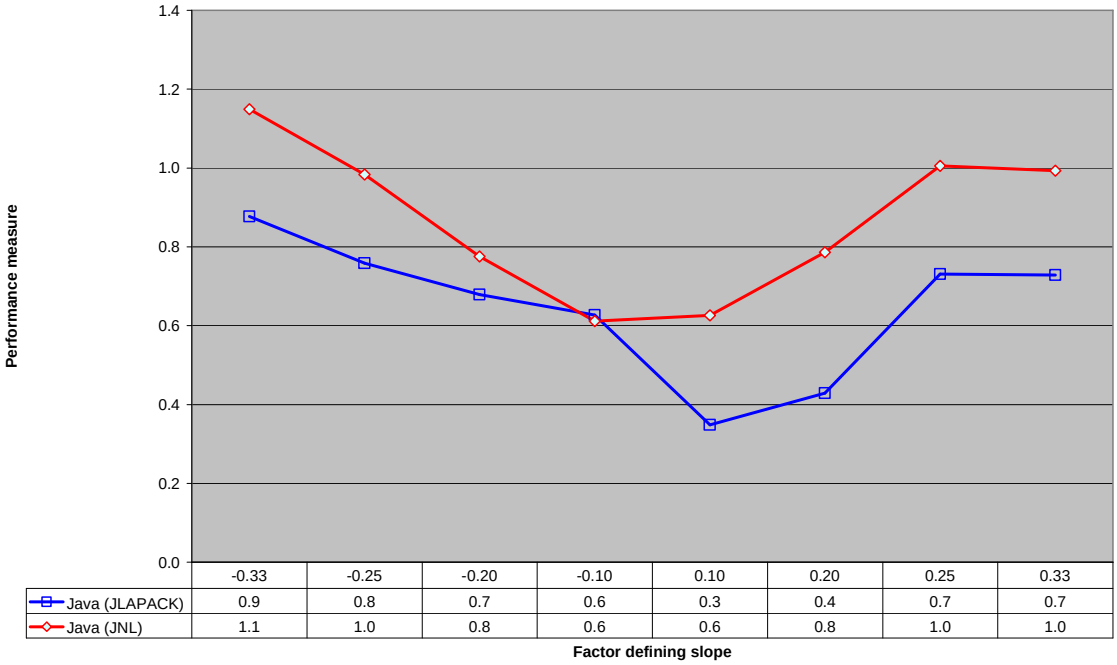
Figure 6: As Figure 4 except that the performance parameter is the sample size.

Appendix D: Results for Ordinary Straight-Line Regression



	1.0E+00	1.0E+01	1.0E+02	1.0E+03	1.0E+04	1.0E+05	1.0E+06	1.0E+07
Java (JLAPACK)	0.7	1.8	2.7	3.6	4.6	5.5	7.1	7.6
Java (JNL)	1.0	1.3	5.4	6.7	8.9	11.1	12.4	15.9

Figure 7: Values of the performance measure for test software for the computation of ordinary straight-line regression. The performance parameter is the coordinate of the centroid of the data *x*-values.



	-0.33	-0.25	-0.20	-0.10	0.10	0.20	0.25	0.33
Java (JLAPACK)	0.9	0.8	0.7	0.6	0.3	0.4	0.7	0.7
Java (JNL)	1.1	1.0	0.8	0.6	0.6	0.8	1.0	1.0

Figure 8: As Figure 7 except that the performance parameter is the factor defining the slope of the solution line.

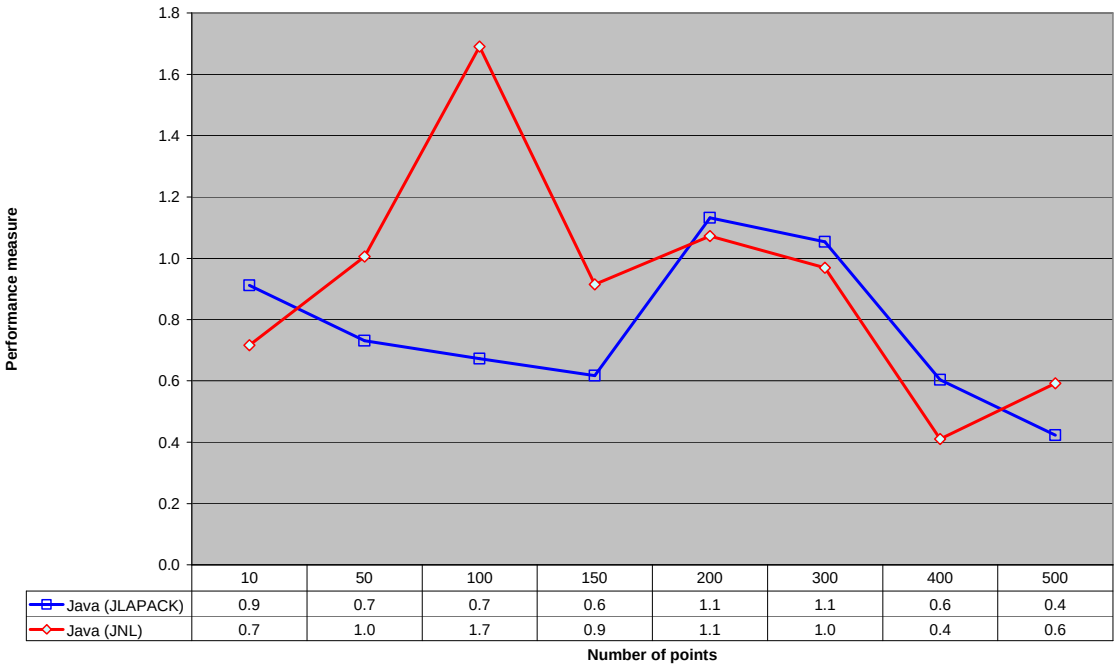


Figure 9: As Figure 7 except that the performance parameter is the number of data points.

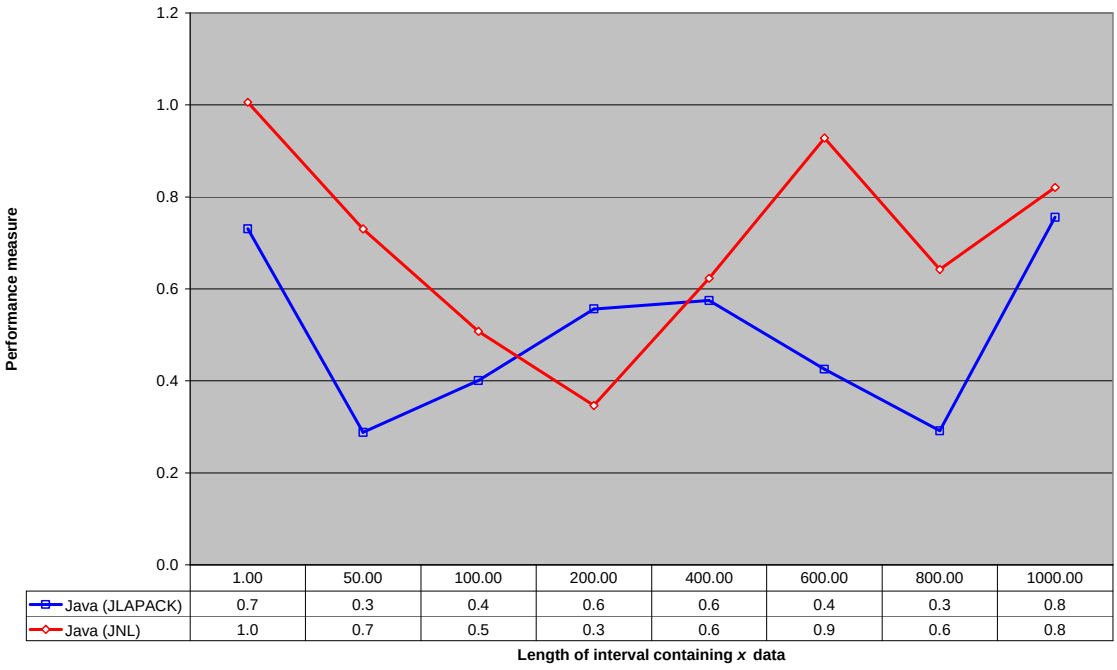


Figure 10: As Figure 7 except that the performance parameter is the length of the interval containing the data *x*-values.

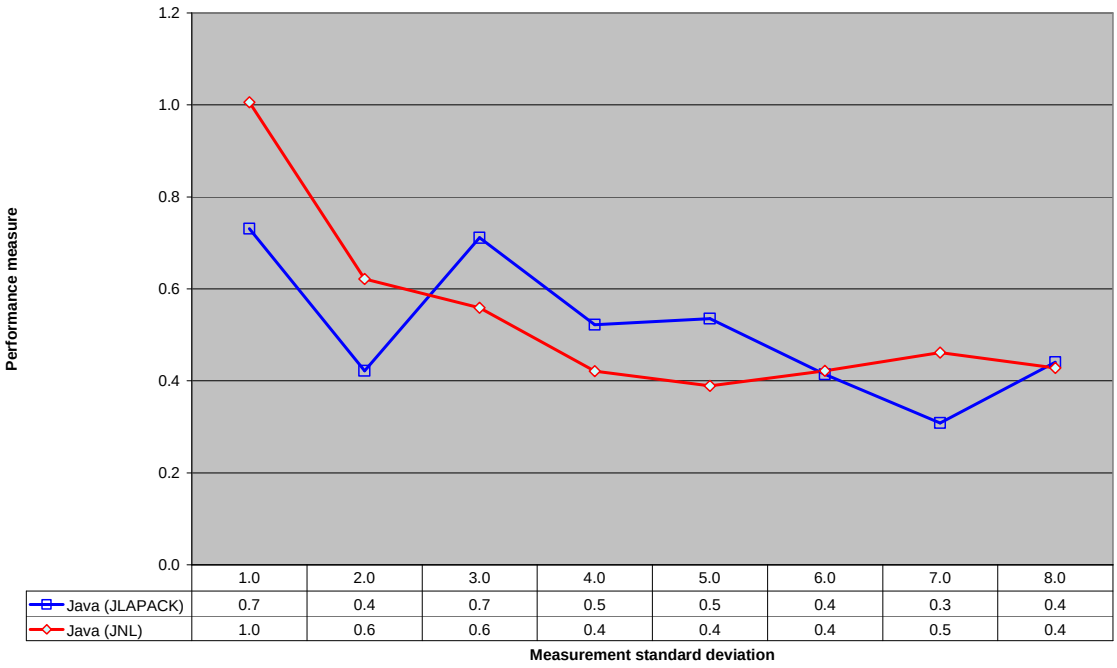


Figure 11: As Figure 7 except that the performance parameter is the measurement standard deviation.

Appendix E: Results for Mathematical Functions

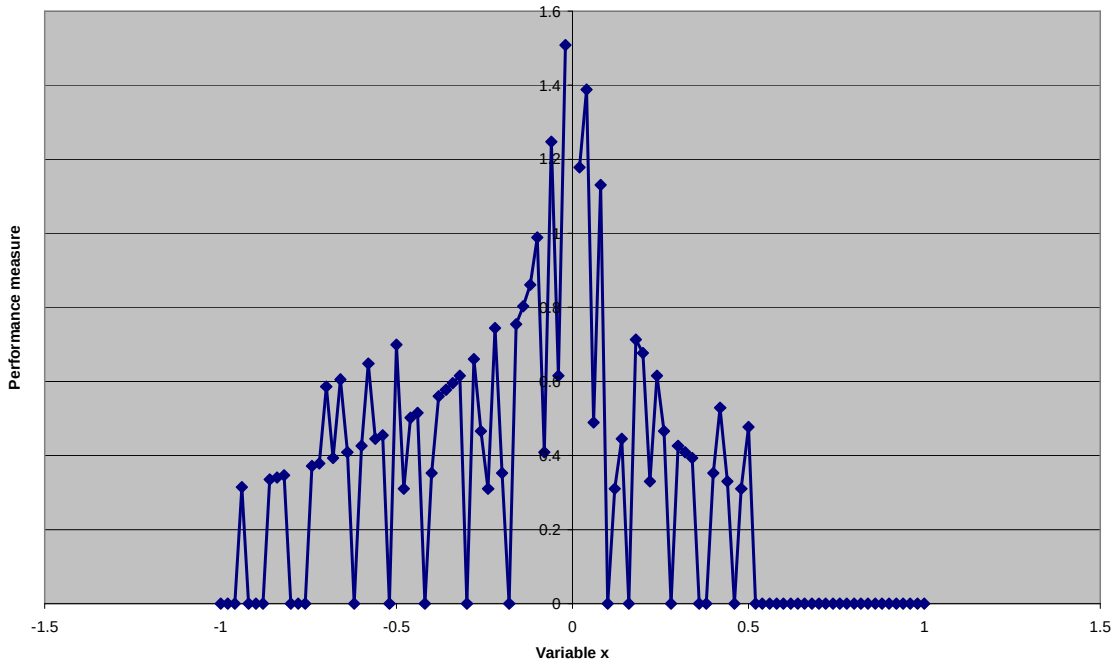


Figure 12: Values of the performance measure for complementary test T3.

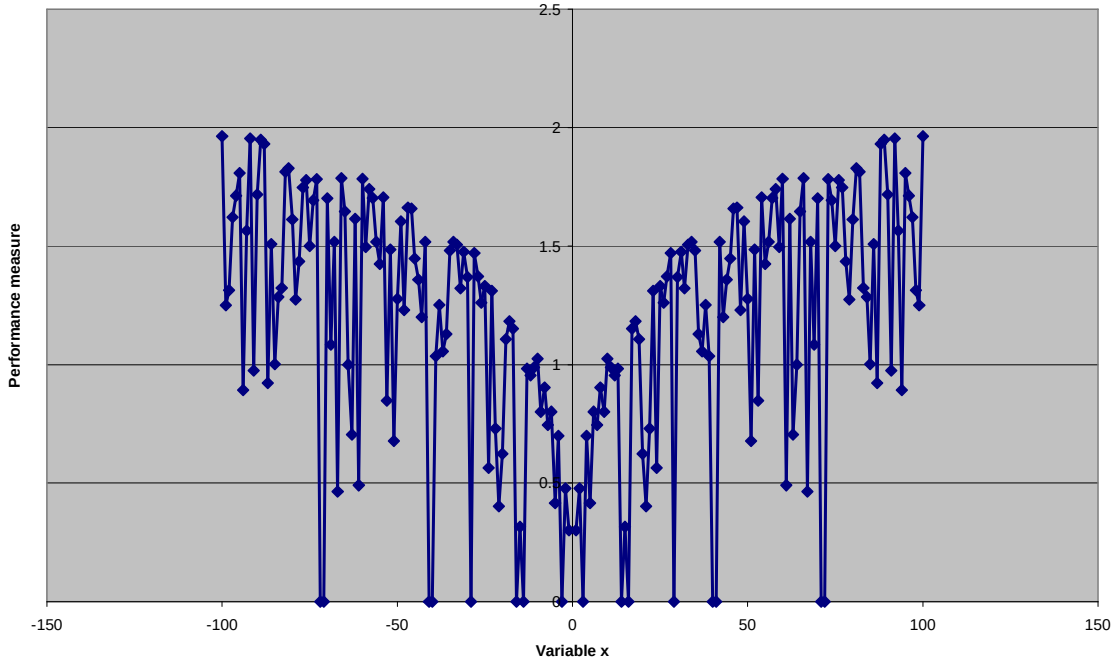


Figure 13: Values of the performance measure for complementary test T5.

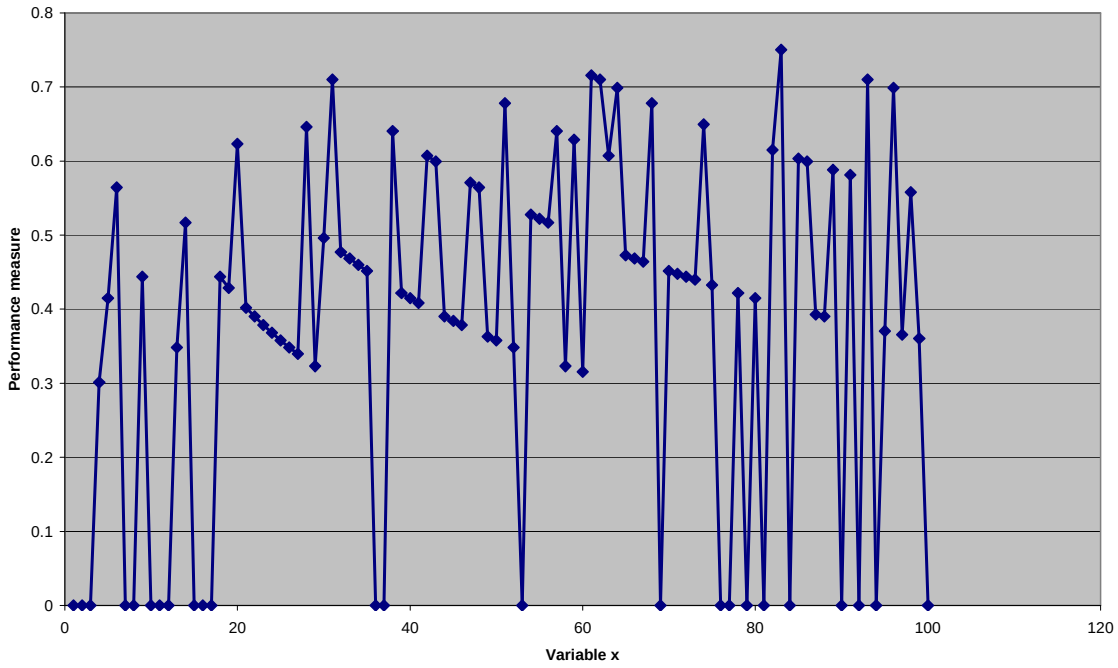


Figure 14: Values of the performance measure for complementary test T10.

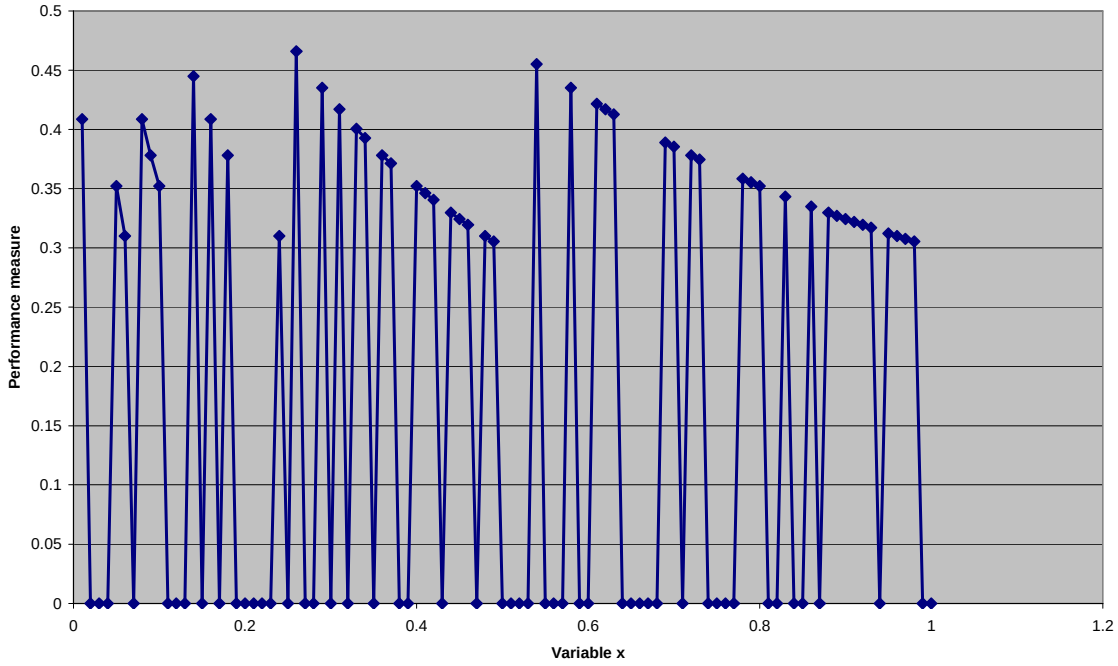


Figure 15: Values of the performance measure for complementary test T11.

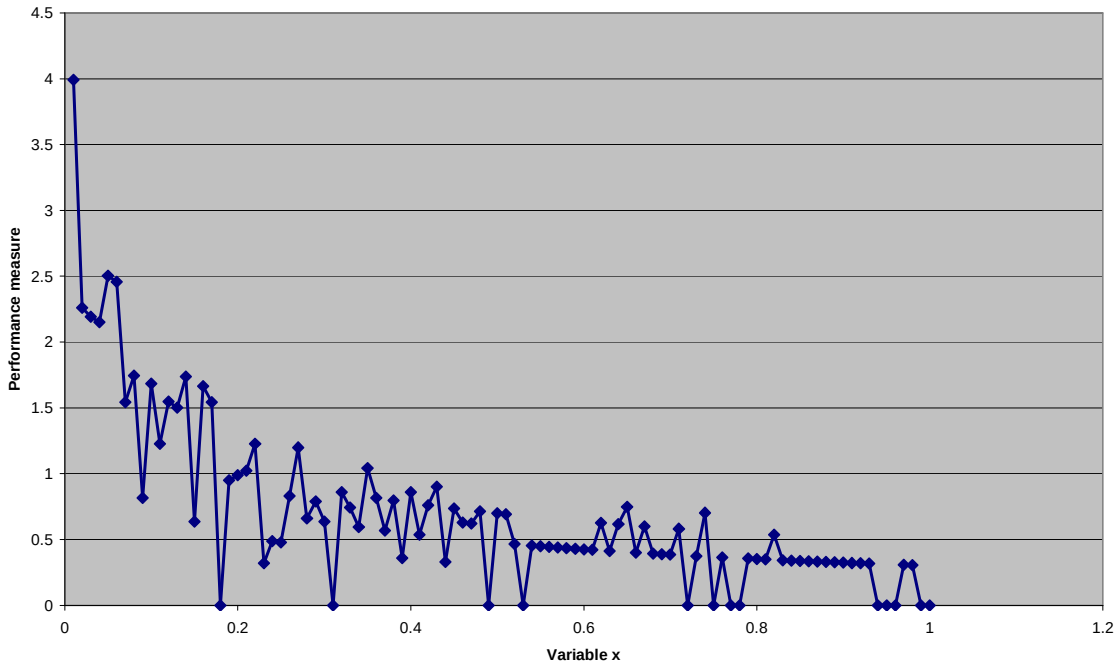


Figure 16: Values of the performance measure for complementary test T13.

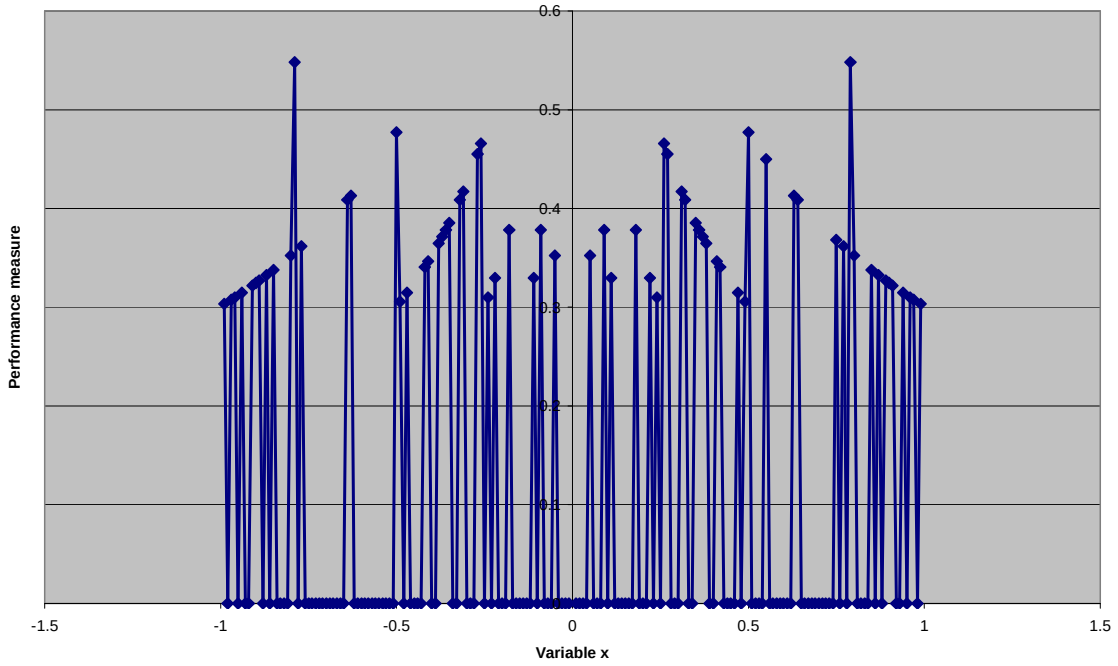


Figure 17: Values of the performance measure for complementary test T14.

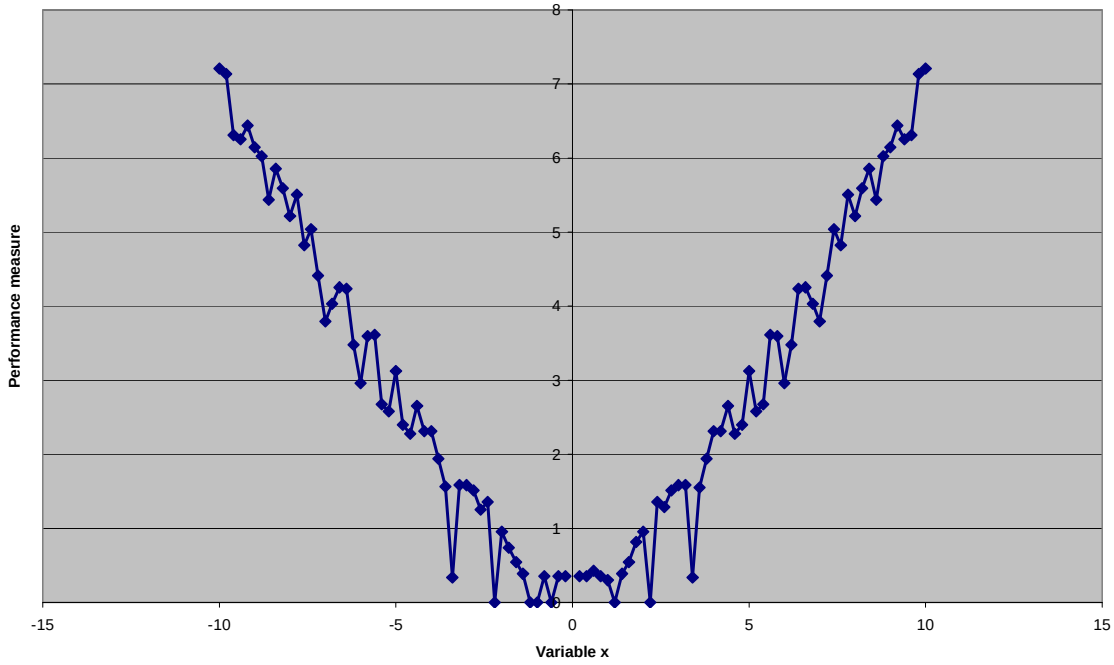


Figure 18: Values of the performance measure for complementary test T15.

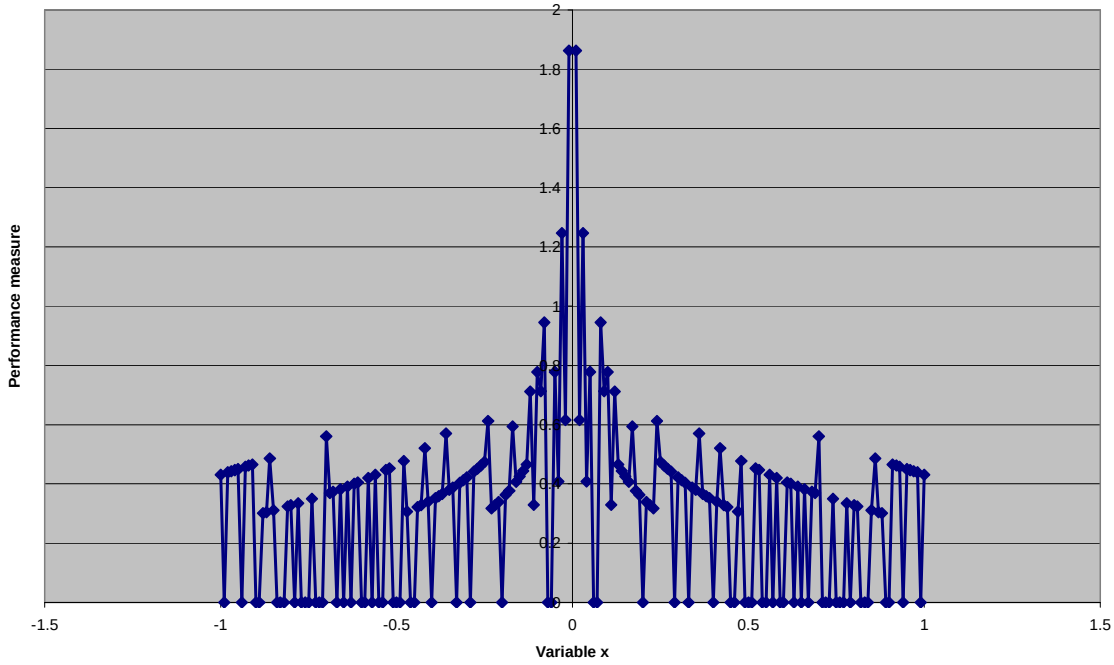


Figure 19: Values of the performance measure for complementary test T18.

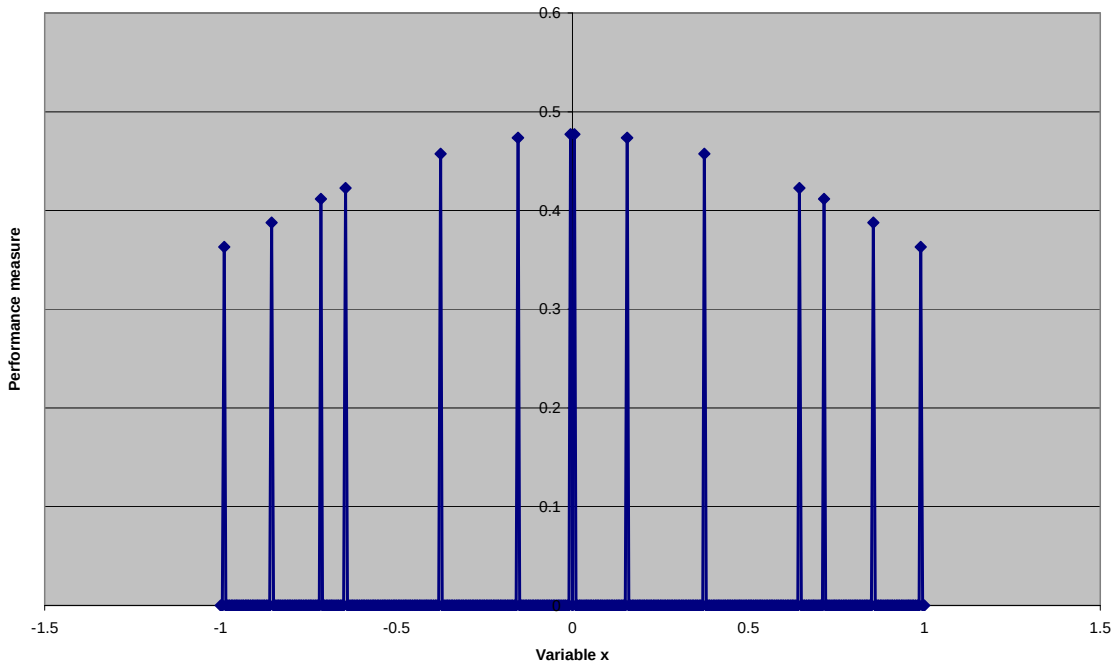


Figure 20: Values of the performance measure for complementary test T19.

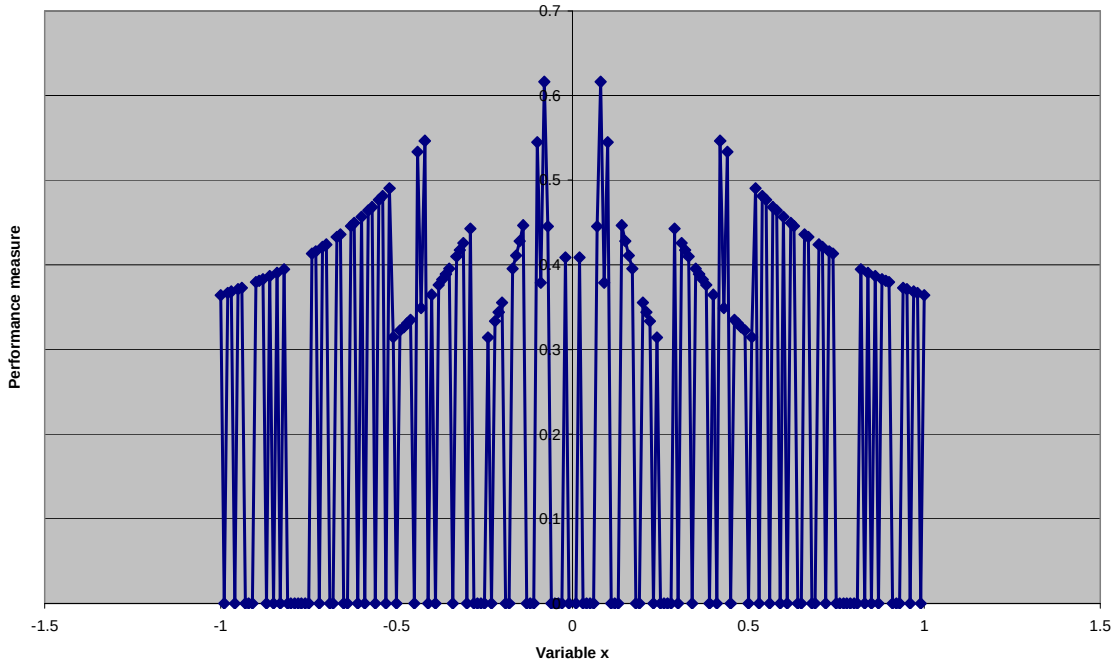


Figure 21: Values of the performance measure for complementary test T20.

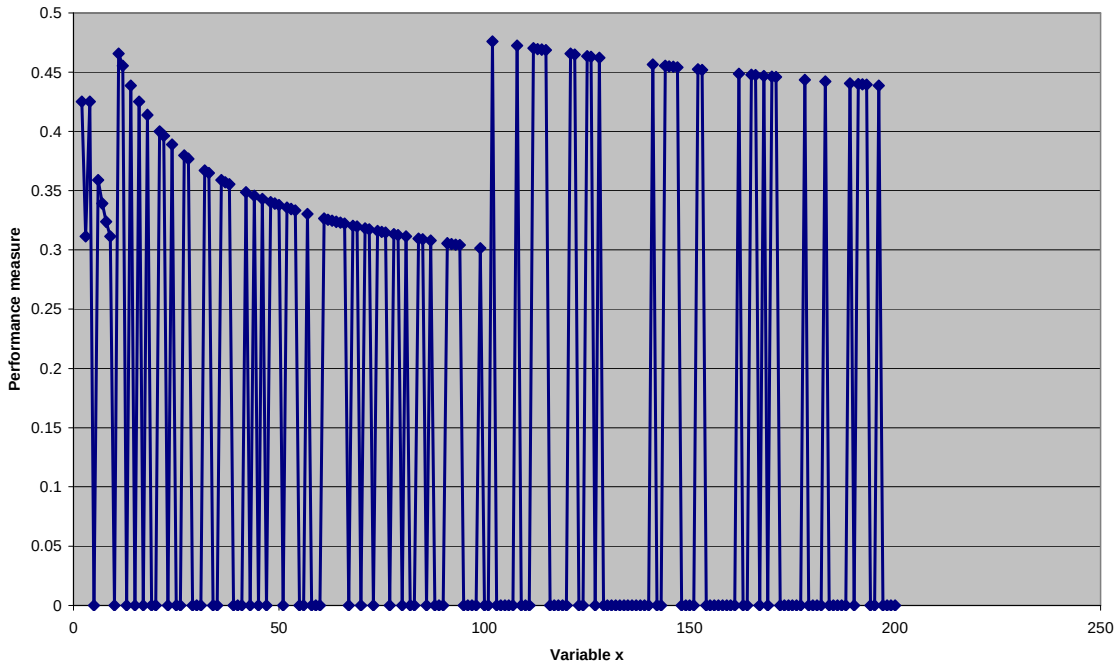


Figure 22: Values of the performance measure for complementary test T23.

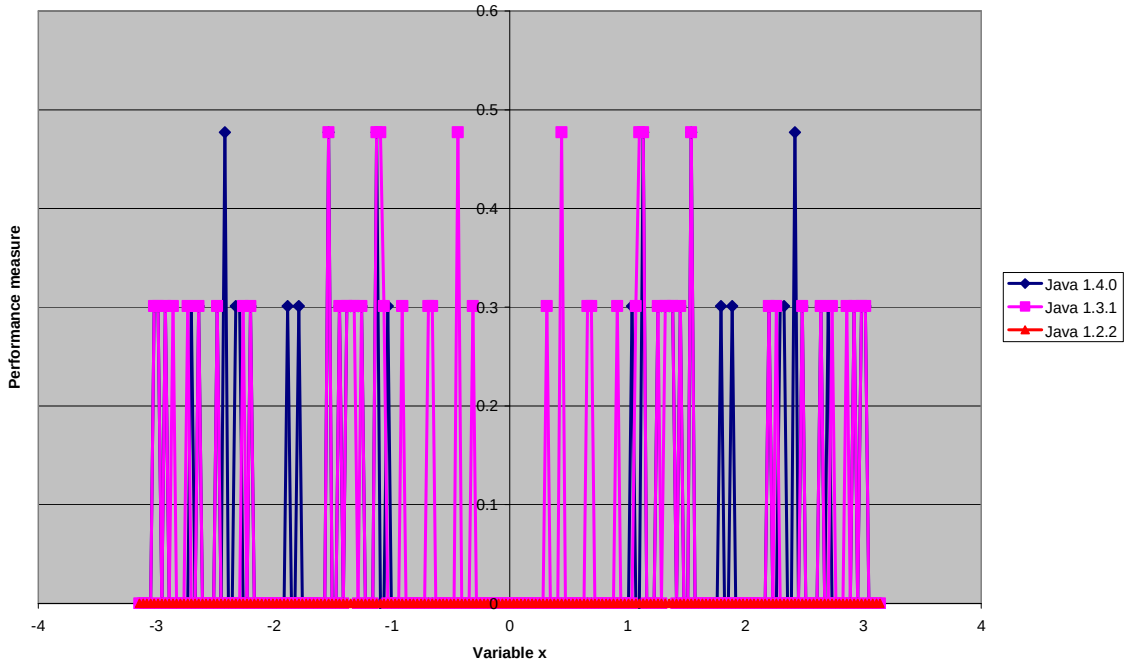


Figure 23: Values of the performance measure for complementary test T16 executed in three different versions of the Java run-time environment.

Appendix F: Results for Statistical Distributions

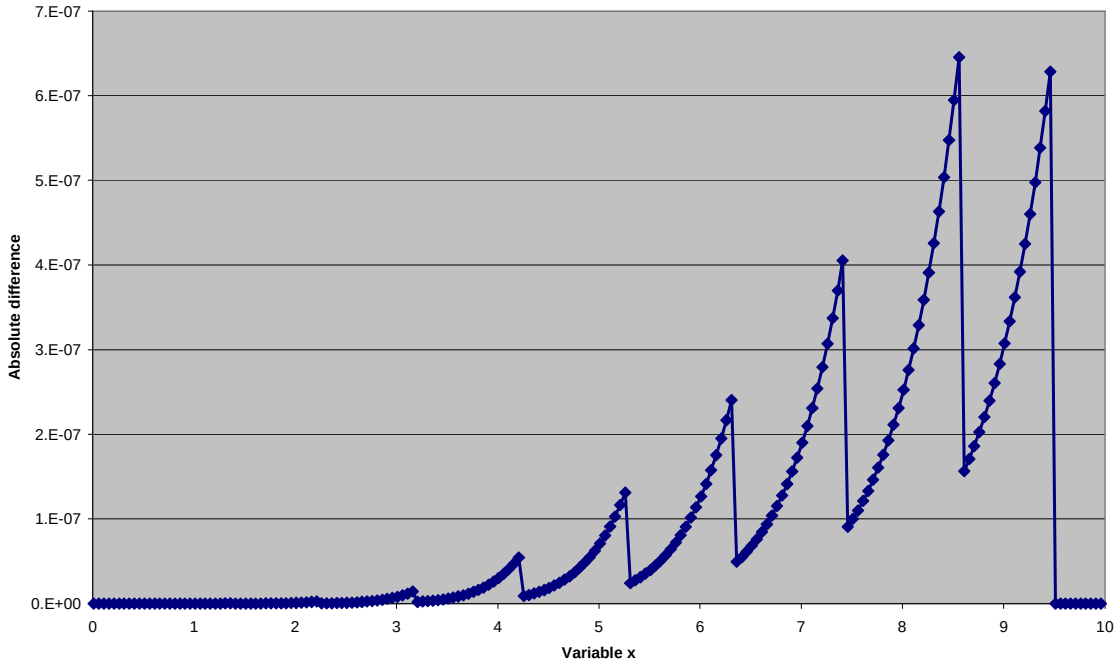


Figure 24: Values of the absolute difference between results from test and reference software for the cumulative χ^2 distribution with ten degrees of freedom.

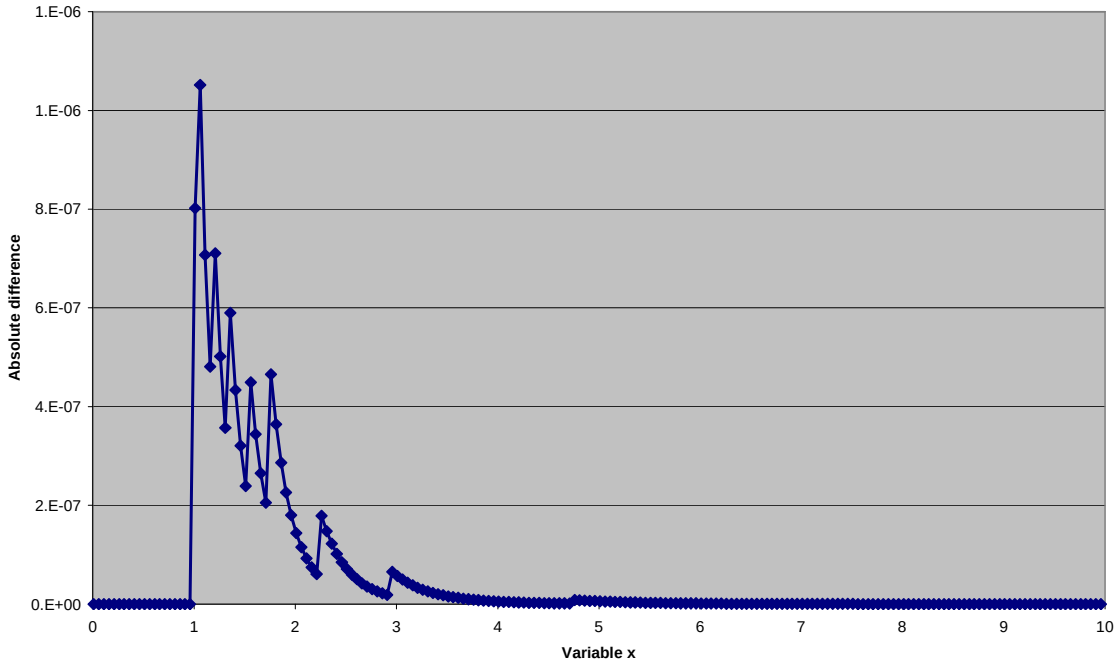


Figure 25: Values of the absolute difference between results from test and reference software for the cumulative *F* distribution with five and ten degrees of freedom.

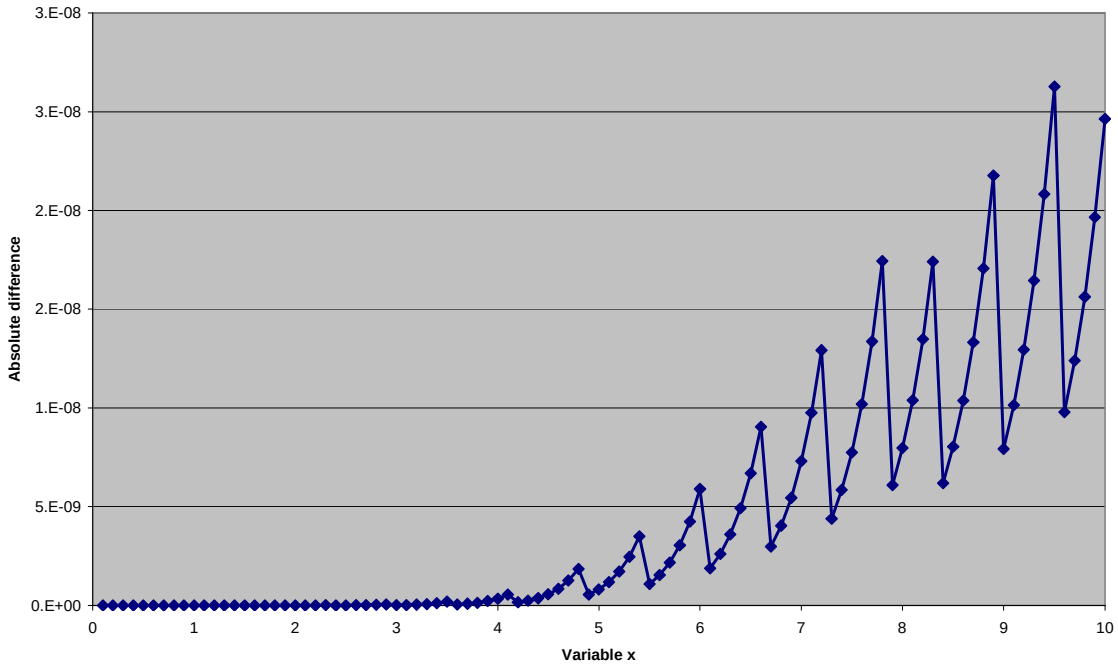


Figure 26: Values of the absolute difference between results from test and reference software for the Poisson distribution.

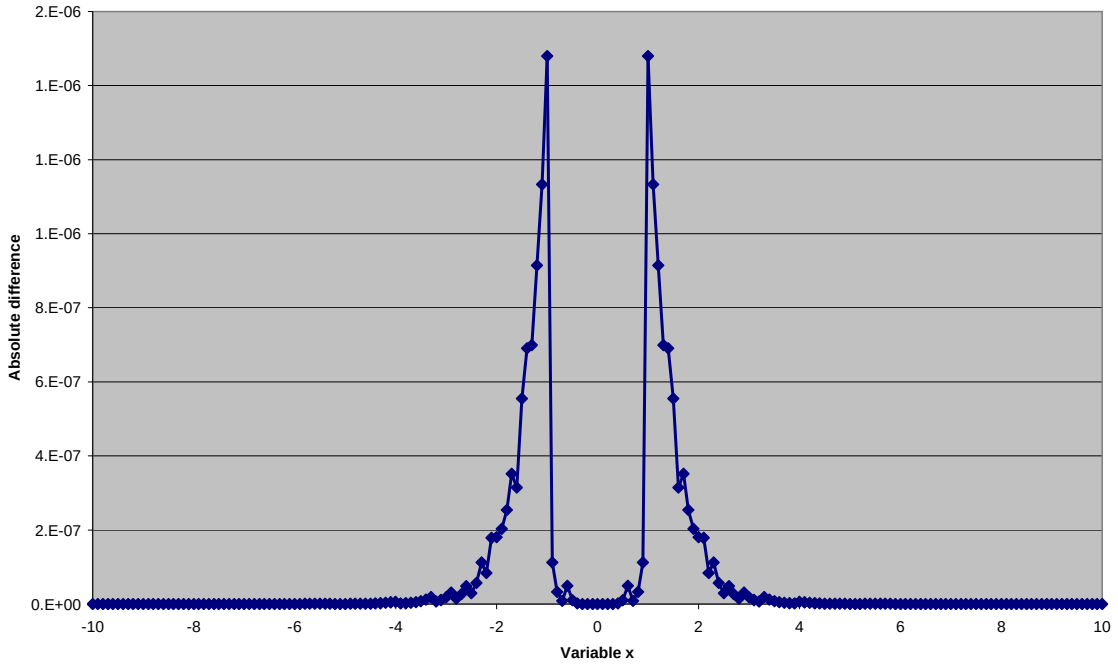


Figure 27: Values of the absolute difference between results from test and reference software for the cumulative Students'-*t* distribution with five degrees of freedom.

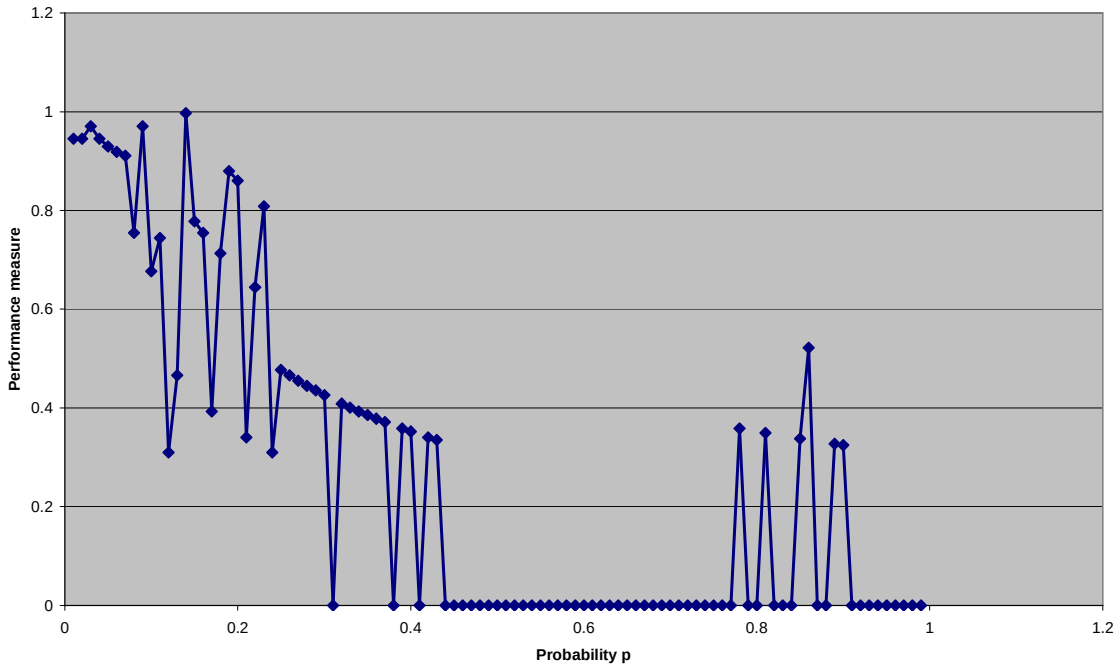


Figure 28: Values of the performance measure for complementary test T24.

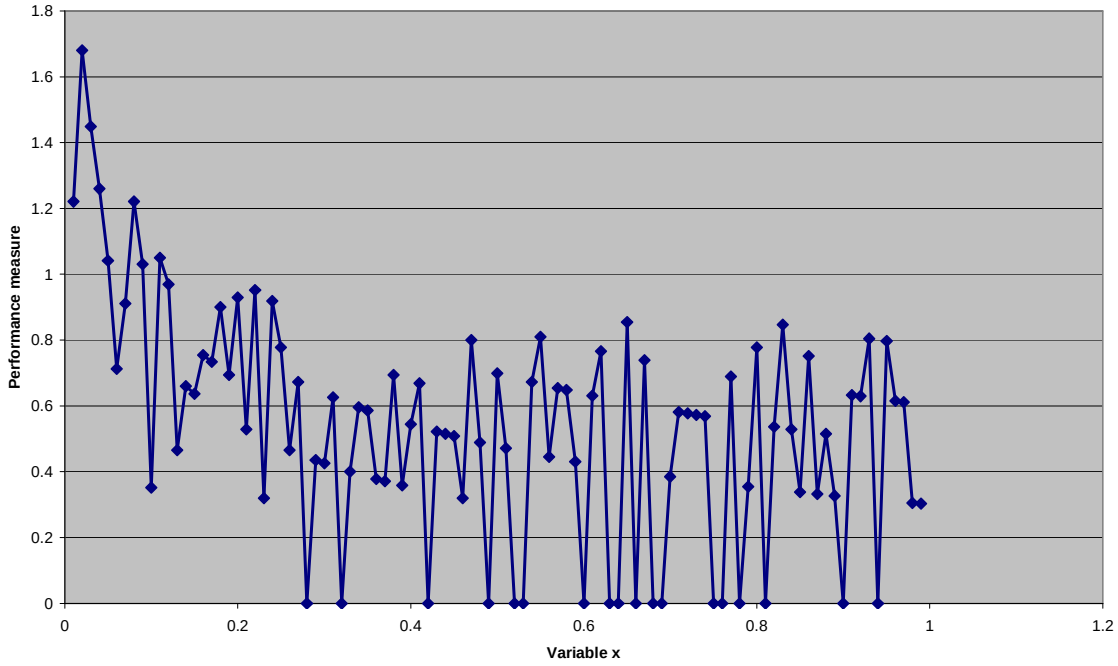


Figure 29: Values of the performance measure for complementary test T25.

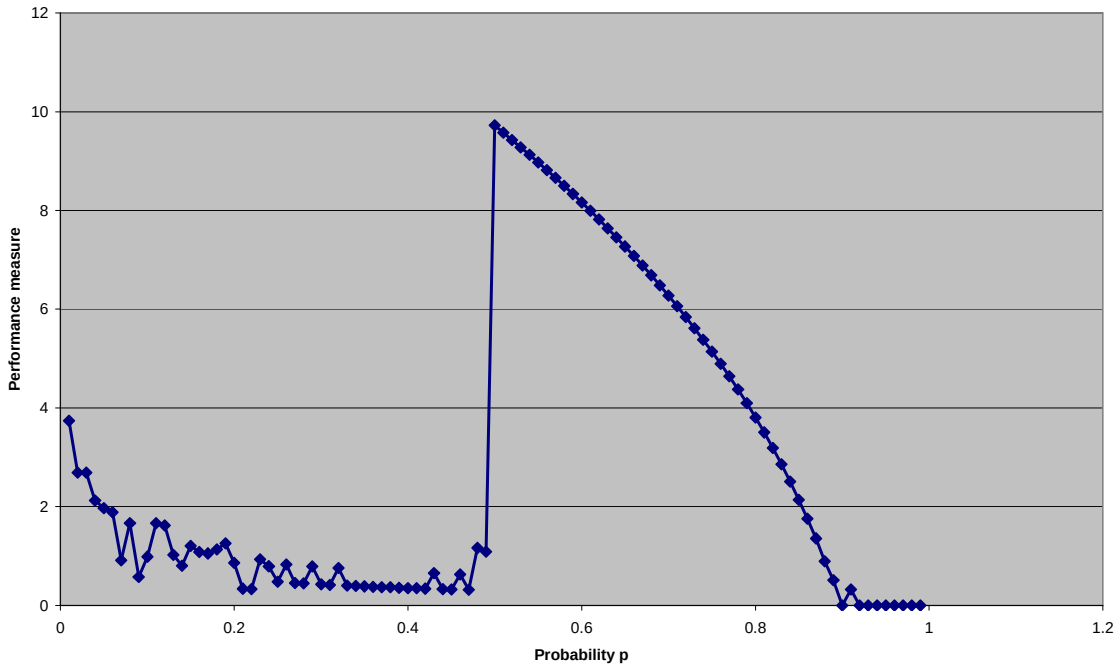


Figure 30: Values of the performance measure for complementary test T26.

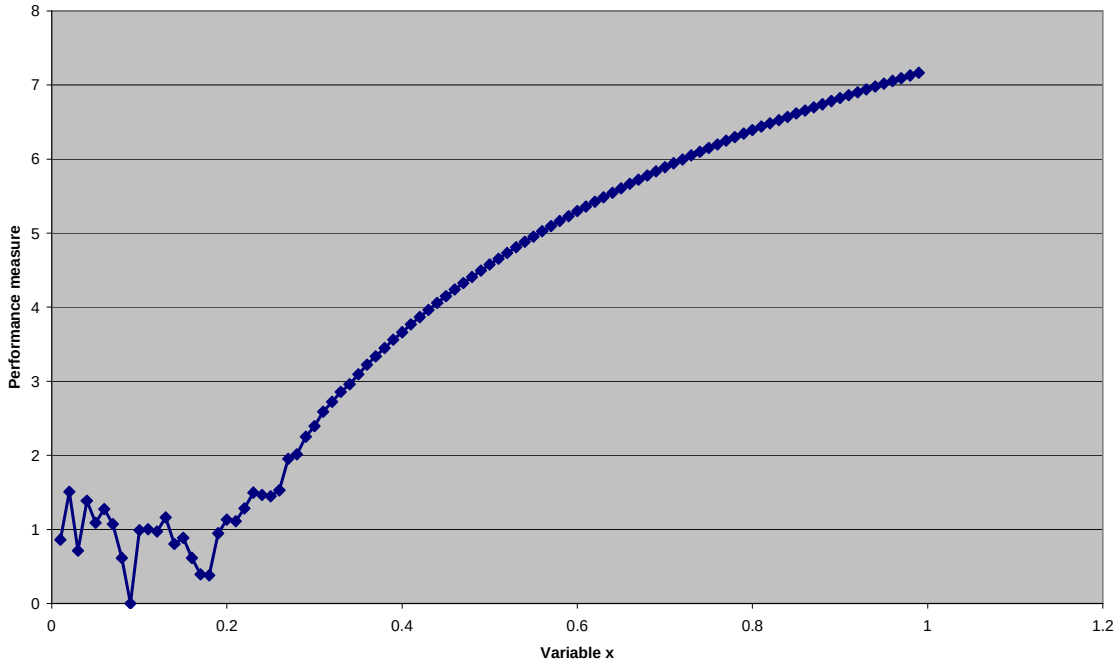


Figure 31: Values of the performance measure for complementary test T27.