NPL Report DITC 211/92

Strategies for testing form assessment software

M G Cox and A B Forbes

September 1999

NPL Report DITC 211/92

September 1999

Strategies for testing form assessment software

M G Cox and A B Forbes
Division of Information Technology and Computing
National Physical Laboratory
Teddington
Middlesex
United Kingdom
TW11 0LW

ABSTRACT

This report describes methods for generating test data sets and reference results which can be used to test assessment software employed in coordinate metrology for finding least-squares best-fit curves and surfaces to data. The methods do not rely on the use of reference software but instead exploit known mathematical properties which hold at the solution of the data approximation problem. The methods are flexible, straightforward to implement and should be of value in designing conformance testing procedures. The methods are illustrated for the problem of finding the least-squares best-fit circle to data.

ISSN 0262-5369

National Physical Laboratory
Teddington, Middlesex, United Kingdom, TW11 0LW

Approved on behalf of Chief Executive, NPL,
by Mr A J Marks, Head, Division of Information Technology and Computing

CONTENTS

Page

1 INTRODUCTION 1

2 MATHEMATICAL FORMULATION OF THE LEAST-SQUARES ELEMENT FITTING PROBLEM 2

3 GENERATION OF DATA SET/REFERENCE RESULT PAIRS 3

3.1 REFERENCE SOFTWARE 3

3.2 REFERENCE BASES 4

3.2.1 Characterisation of a local minimum 5
3.2.2 Perturbations normal to the element 5
3.2.3 Summary of data set generation 7

3.3 MINIMAL PERTURBATIONS AND FORM ERROR 9

3.4 VALIDATION OF NULL SPACE METHODS 11

3.5 GENERALISATIONS 11

4 INVARIANCE TESTING 12

5 COMPARING BEST-FIT ELEMENT RESULTS 12

5.1 COMPARISON BASED ON PARAMETER VALUES 12

5.2 COMPARISON BASED ON RESIDUAL ERRORS 13

6 RESIDUAL ERRORS, REFERENCE SOFTWARE AND PARAMETER DETERMINATION 13

7 DESIGN OF TEST DATA SETS 14

8 PRACTICAL CONFORMANCE TESTING 15

8.1 BASIC PROCEDURE 15

8.2 COMPUTER INTENSIVE SOFTWARE TESTING. 15

9 CASE STUDY: CIRCLE ALGORITHMS 16

9.1 TEST RESULTS ON DATA SETS X_1, X_2, X_3 AND X_4 17

9.2 TRANSLATION INVARIANCE TESTING 18

9.3 RECOVERY OF PARAMETER VALUES FROM RESIDUAL ERRORS . . . 19

9.4 DISCUSSION 20

10 SUMMARY AND CONCLUSIONS 21

11 ACKNOWLEDGEMENTS 21

12 REFERENCES 21

APPENDICES

APPENDIX 1 Four reference pairs for circle fitting. 23

1 INTRODUCTION

This report is concerned with the generation and use of reference data sets for testing assessment software. Assessment software plays a fundamental part in coordinate metrology which in turn is a key component of computer-aided inspection in manufacturing. In order to verify that a manufactured artefact is within tolerance a coordinate measuring machine (CMM) will collect a set of data representing points lying on the surface of the artefact and assessment software will compare these data points with the design and tolerance specification for the artefact [16]. It is clearly important that such assessment software gives reliable answers and any serious quality management system will demand that its reliability can be verified in a practical manner. Given an explicit description of its computational aim, software can be tested using the following approach [8]:

I Determine reference data sets (appropriate for the computational aim) and corresponding reference results;

II Apply the software under test to the reference data sets to produce test results;

III Compare the test results with reference results.

This approach poses two fundamental questions:

Q1 How can we determine correct reference results for data sets?

Q2 How can we compare test results with reference results in a meaningful way?

The viability of the approach depends on satisfactory answers to these questions and no formal software conformance testing will be worthwhile unless these questions have been addressed.

Software packages for finding the least-squares best-fit geometric elements – lines, planes, circle, spheres, cylinders and cones – are among the most commonly used in coordinate metrology. In this report, we describe a number of methods which, when taken together, ensure that testing element-fitting software using data sets can have a firm foundation. Furthermore, the methods can be applied to testing more general curve and surface regression software.

The remainder of this report is organised as follows. In section 2, we give a mathematical formulation of the element-fitting problem. In section 3, we show how an analysis of the fitting problem leads to methods for simultaneously generating data sets and reference results, and in section 4 we look at other methods for testing software which are also based on mathematical properties of the problem. The problem of how to compare test results is discussed in sections 5 and 6. We consider the design of test data sets in section 7 and practical conformance testing procedures in section 8. Much of the discussion is illustrated in the case study presented in section 9. Our concluding remarks are set out in section 10.

[0]This document: [abf.tex.swt]swt.ps.

2 MATHEMATICAL FORMULATION OF THE LEAST-SQUARES ELEMENT FITTING PROBLEM

Given a data set $X = \{\mathbf{x}_i : i \in I\}$ and an element $\mathcal{E}$, the least-squares error of fit D of $\mathcal{E}$ at X is defined by

$$D(X;\mathcal{E}) = \sum_{i \in I} d_i^2, \tag{1}$$

where $d_i = d(\mathbf{x}_i;\mathcal{E})$ is the distance from the point $\mathbf{x}_i$ to $\mathcal{E}$. The least-squares best-fit element, if it exists, is the element $\mathcal{E}^*$ which minimises $D(X;\mathcal{E})$ among all instances $\mathcal{E}$ of the element. If the element is parametrized by parameters $\mathbf{u} = (u_1, \ldots, u_n)$ so that to each $\mathbf{u}$ we associate a unique element $\mathcal{E}(\mathbf{u})$, then the distance d from a point $\mathbf{x}$ to $\mathcal{E}(\mathbf{u})$ can be written as a function $d(\mathbf{x};\mathbf{u})$ of $\mathbf{x}$ and $\mathbf{u}$. Similarly, given a data set X, the error of fit D can be written as a function of $\mathbf{u}$ (cf. (1)):

$$D(X;\mathbf{u}) = \sum_{i \in I} d^2(\mathbf{x}_i;\mathbf{u}), \tag{2}$$

and the element-fitting problem can be posed as

$$\min_{\mathbf{u}} D(X;\mathbf{u}). \tag{3}$$

Methods of parametrizing geometric elements are considered in some detail in [2]. As explained therein, no one parametrization can describe all axis directions[1] so that for elements like the cylinder and cone more than one parametrization is required to specify all instances of the element. It follows that (3) is well-posed only if the optimal $\mathcal{E}^*$ can be specified by $\mathbf{u} \mapsto \mathcal{E}(\mathbf{u})$, i.e., there exists $\mathbf{u}^*$ such that $\mathcal{E}^* = \mathcal{E}(\mathbf{u}^*)$. In the following discussion, we assume that this is the case.

An algorithm for finding a best-fit element to a set X of data points has three ingredients: i) a parametrization $\mathbf{u} \mapsto \mathcal{E}(\mathbf{u})$ of the element, ii) a definition of an error function $E(X;\mathbf{u})$ to be minimised with respect to $\mathbf{u}$, and iii) a method to produce an estimate $\hat{\mathbf{u}}$ of the point $\mathbf{u}^*$ at which E takes its minimum.

Example: Least-squares best-fit circle. A circle in the plane can be parametrized by its centre coordinates (a, b) and its radius r_0. The distance d from a point $\mathbf{x} = (x, y)$ to a circle defined by parameters $\mathbf{u} = (a, b, r_0)$ is given by

$$d = d(\mathbf{x};\mathbf{u}) = \left[(x-a)^2 + (y-b)^2\right]^{1/2} - r_0. \tag{4}$$

Given a set of data points $X = \{\mathbf{x}_i : i \in I\}$, the associated least-squares error function D is given by (2) and can be minimised by using the Gauss-Newton algorithm, for example; see [12, 17]. □

A comprehensive evaluation of element-fitting software should test all three aspects of the underlying algorithm: it should find the circumstances under which the parametrization $\mathbf{u} \mapsto \mathcal{E}(\mathbf{u})$ is inappropriate, investigate to what extent the minimum of the error function $E(X;\mathbf{u})$ coincides with that of $D(X;\mathcal{E}(\mathbf{u}))$ and examine the effectiveness of the method of finding the minimum of the error function. However, before we can tackle the problem of how to design a suite of test data sets X_q which can be used to analyse thoroughly the behaviour of assessment software, it is necessary to address the two questions raised in the introduction.

[1] The fundamental reason for this is that there is no one-to-one continuous mapping from the plane onto the sphere.

3 GENERATION OF DATA SET/REFERENCE RESULT PAIRS

The use of test data sets relies on the existence of reference results $\mathbf{r}$ for a data set X. We refer to $\langle X, \mathbf{r} \rangle$ as a *reference pair*. In this section, we consider the question of how to arrive at reference results $\mathbf{r}$.

3.1 REFERENCE SOFTWARE

A first response to Q1 is to design 'reference' software R to determine reference results $\mathbf{r} = R(X)$ for a given data set X. Before going further, it may be appropriate to indicate what is meant by reference software in this context.

We can only speak of reference software if we have a clear, unambiguous statement of the computational aim of the software. In most data fitting applications, this aim can normally be expressed in mathematical notation, for example: Given a data set $X = \{\mathbf{x}_i : i \in I\}$ and error function $E = E(X, \mathbf{u})$ depending on the data X and parameters $\mathbf{u} = (u_1, \ldots, u_n)$ find the $\mathbf{u}^*$ which minimises E.

Once a computational aim has been established there remains the problem of how to check that a solution output by a piece of software has satisfied the computational aim. This in general is a non-trivial problem and is usually only partially solved by deriving conditions which must hold at the solution and examining if these conditions are satisfied.

Given an explicit computational aim and the task of writing software to achieve that aim, there will be a number of factors to consider, among them:

Correctness: will the software produce the required solution?

Speed: how long can the computation take?

Memory: how much space can the software use?

Robustness: should the software work even for metrologically unrepresentative data?

Generality: should the software treat the problem in full generality or can it make extra assumptions about the problem?

Maintainability.

Usability: how expert is the user?

For complicated computational tasks there is likely to be conflict for example between correctness, robustness and generality on the one hand and speed and memory on the other. For reference software, the emphasis is on correctness, robustness and generality. For production software the emphasis may be on speed and memory. However extreme positions on the spectrum will not in general be acceptable: reference software should not be hopelessly slow, production software should not be wildly inaccurate.

If the computational task $\min_{\mathbf{u}} E(X, \mathbf{u})$ is too difficult to solve within the practical constraints there are a number of options which will yield approximate solutions. For example, an iterative algorithm employed to find the minimum $\mathbf{u}^*$ can be stopped after a fixed number of iterations with output $\tilde{\mathbf{u}}$; or the error function $E(X, \mathbf{u})$ can be replaced by one $\bar{E}(Y, \mathbf{v})$ whose minimiser

$\bar{\mathbf{v}}^*$ is easier to calculate. Both approaches are common in practice and their validity depends on how close the approximate solutions so generated are to the true solution $\mathbf{u}^*$. It is often possible to make theoretical statements under certain assumptions about the data X, but verifying that these assumptions hold in practice may be difficult.

Our working definition of reference software is software that addresses the exact computational aim using numerically stable techniques.

Reference software can be used in two ways:

Reference software as production software. If it is possible to address correctly the computational aim and also satisfy the constraints for production software, then reference software, if required, can be implemented as production software. In this fashion, the construction of reference software is a means of *directly* improving the software used in everyday calculations. For this use the software should be designed in such a fashion so as to be embeddable in other systems. In particular the software should be written in terms of standard computational modules where possible.

Generation of test data. If reference software does not satisfy the constraints for production software, then it can be used to test the validity of methods implemented as production software. This can done by generating test data and using reference software to calculate the corresponding reference results. In this way, reference software can *indirectly* improve production software.

Returning to the element-fitting problem, we can employ stable parametrizations [2], optimisation algorithms with proven convergence characteristics [12, 17] and reliable linear algebra modules with excellent error behaviour [18, 28] along with an exact mathematical modelling of the element fitting problem [14], to develop reference software in which we can have great confidence. However, there are a number of reasons why it is important to test element-fitting software in a manner which does not rely on the existence of reference software: firstly, there is (at present) no practical guarantee that a software implementation of a reliable algorithm is error-free and will *always* perform satisfactorilyy;[(2)] secondly, the development of reference software for non-trivial tasks requires significant resources; thirdly, the problem of testing the reference software has to be resolved.

3.2 REFERENCE BASES

In this section we describe a method which generates data sets and reference results *simultaneously* in a way that obviates the need for reference software.

We have seen that the element-fitting problem is: Given a data set X find the minimum $\mathbf{u}^*$ of the error function $D(X;\mathbf{u})$. The approach to generating reference pairs addresses a 'dual' problem: Given $\mathbf{u}^*$, find a data set X for which $D(X;\mathbf{u})$ takes its minimum value at $\mathbf{u}^*$. We solve this problem by first examining the conditions for $\mathbf{u}$ to be a (local) minimum of $D(X;\mathbf{u})$.

[(2)]However, *formal methods* for developing (and testing) software, given a formal specification of requirements, have this as their goal; see for example [23, 22]. Unfortunately, these methods do not apply to computations involving floating point operations and rounding error.

3.2.1 Characterisation of a local minimum

See, for example, [12, 17].

Let $\mathbf{u} \mapsto \mathcal{E}(\mathbf{u})$ be a parametrization of an element $\mathcal{E}$, and let $d(\mathbf{x}; \mathbf{u})$ describe the distance from a point $\mathbf{x}$ to $\mathcal{E}(\mathbf{u})$ in terms of the parameters $\mathbf{u}$. Given a set of points $X = \{\mathbf{x}_i : i \in I\}$, let $\mathbf{d}$ be the column vector whose ith component is

$$d_i = d(\mathbf{x}_i; \mathbf{u}),$$

and define the *Jacobian* matrix by

$$J_{ij} = \frac{\partial d_i}{\partial u_j}.$$

Set G_i to be equal to the matrix of second partial derivatives corresponding to the ith point:

$$G_i(j,k) = \frac{\partial^2 d_i}{\partial u_j \partial u_k}.$$

Necessary conditions for $\mathbf{u}$ to be a local minimum of the least-squares objective function

$$D(X, \mathbf{u}) = \sum_{i \in I} d^2(\mathbf{x}_i; \mathbf{u}) \tag{5}$$

are that i)

$$J^{\mathrm{T}}\mathbf{d} = \mathbf{0}, \tag{6}$$

(so that $\mathbf{d}$ lies in the *null space* of J^{T}) and ii) the *Hessian* matrix

$$H = J^{\mathrm{T}}J + \sum_{i \in I} d_i G_i \tag{7}$$

is positive semi-definite (all eigenvalues non-negative).

Sufficient conditions for $\mathbf{u}$ to be a local minimum are that (6) holds and that the Hessian matrix H is strictly positive definite (all eigenvalues strictly greater than zero).

3.2.2 Perturbations normal to the element

Suppose $X^* = \{\mathbf{x}_i^* : i \in I\}$ is a set of points lying exactly on the element $\mathcal{E}(\mathbf{u}^*)$ and J^* is the associated Jacobian matrix. Note that for this set $\mathbf{d} = \mathbf{0}$ so that the first sufficiency condition (6) holds. The second sufficiency condition will hold if J^* is of full rank for then $H = {J^*}^{\mathrm{T}}J^*$ will be strictly positive definite. The rank of J^* depends on the parametrization $\mathbf{u} \mapsto \mathcal{E}(\mathbf{u})$ and the number and distribution of the points X^*. We assume that the parametrization is non-singular near $\mathbf{u}^*$ which means in essence that a change in the parameter values results in a change in the position of the element's surface. We also assume that the points in X^* are sufficient in number and distributed so that $\mathcal{E}(\mathbf{u}^*)$ is uniquely determined (locally) by the condition that the element passes through the points in X^* [3]. Under these assumptions, J^* will be of full rank.

Given such an X^*, we look for a perturbed set $X = \{\mathbf{x}_i : \mathbf{x}_i = \mathbf{x}_i^* + \mathbf{e}_i,\ i \in I\}$ such that the sufficiency conditions remain satisfied. In general, the Jacobian matrix associated with the

[3] For example, four points in general position determine a sphere. However, four points lying on a circle in three dimensions only constrains the sphere centre to lie on a straight line.

perturbed set X will be different from J^*. We now show that perturbations normal to the surface do not change the Jacobian.

Given a point $\mathbf{x}$ and an element $\mathbf{u} \mapsto \mathcal{E}(\mathbf{u})$ let $\mathbf{x}^*$ denote the point on $\mathcal{E}(\mathbf{u})$ closest to $\mathbf{x}$. If $\mathbf{x}$ is sufficiently[4] close to $\mathcal{E}(\mathbf{u})$ then $\mathbf{x}^* = \mathbf{x}^*(\mathbf{u})$ is (locally) a well-defined function of $\mathbf{u}$ and the distance function d can be written as

$$d(\mathbf{x}; \mathbf{u}) = \|\mathbf{x} - \mathbf{x}^*(\mathbf{u})\|.$$

Differentiating d with respect to a parameter u_j gives

$$\begin{aligned} \frac{\partial d}{\partial u_j} &= -\frac{\partial \mathbf{x}^{*\mathrm{T}}}{\partial u_j} \frac{\mathbf{x} - \mathbf{x}^*}{\|\mathbf{x} - \mathbf{x}^*\|}, \\ &= -\frac{\partial \mathbf{x}^{*\mathrm{T}}}{\partial u_j} \mathbf{n}^*, \end{aligned}$$

where $\mathbf{n}^*$ is the outward unit normal to the surface at $\mathbf{x}^*$. These derivatives depend only on the direction of $\mathbf{x} - \mathbf{x}^*$ and not on the magnitude. Thus perturbing a point normal to the surface (changing only the magnitude) does not change the derivatives of the distance function. This means that perturbed sets $X_{\boldsymbol{\delta}}$ of the form

$$X_{\boldsymbol{\delta}} = \{\mathbf{x}_i : \mathbf{x}_i = \mathbf{x}_i^* + \delta_i \mathbf{n}_i^*,\ i \in I\},$$

where $\mathbf{n}_i^*$ is the outward unit normal at $\mathbf{x}_i^*$, will also have J^* as the associated Jacobian matrix. Furthermore, the distance from $\mathbf{x}_i = \mathbf{x}_i^* + \delta_i \mathbf{n}_i^*$ to $\mathcal{E}(\mathbf{u}^*)$ is δ_i. Therefore, the sufficiency conditions for $\mathbf{u}^*$ to be a local minimum for the data set $X_{\boldsymbol{\delta}}$ translate as i)

$$J^{*\mathrm{T}} \boldsymbol{\delta} = \mathbf{0}, \tag{8}$$

and ii)

$$H_{\boldsymbol{\delta}} = J^{*\mathrm{T}} J^* + \sum_{i \in I} \delta_i G_{\delta_i} \tag{9}$$

is strictly positive definite, where

$$G_{\delta_i}(j,k) = \frac{\partial^2}{\partial u_j \partial u_k} d_i(\mathbf{x}^* + \delta_i \mathbf{n}_i^*, \mathbf{u}^*).$$

The first condition will be satisfied if $\boldsymbol{\delta}$ lies in the null space Z^* of $J^{*\mathrm{T}}$. Z^* can be calculated from the QR decomposition of J^*; see, e.g. [18, Chapter 6]. If J^* is full rank of dimension $m \times n$, $m > n$, then Z^* is spanned by $m - n$ column vectors $\mathbf{z}_k$, say, $k = 1, \ldots, m - n$, of length m. Any vector $\boldsymbol{\delta}$ satisfying (8) can be written as a linear combination of these vectors,

$$\boldsymbol{\delta} = \sum_{k=1}^{m-n} \nu_k \mathbf{z}_k,$$

and *vice versa*.

Given Z^*, it remains to choose the coefficients ν_k so that the resulting Hessian matrix $H_{\boldsymbol{\delta}}$ is strictly positive definite. If the matrix J^* is full rank then $H_{\boldsymbol{\delta}}$ will be strictly positive definite if the norm of

$$\sum_{i \in I} \delta_i G_{\delta_i}$$

is sufficiently small compared with the minimum eigenvalue of $J^{*\mathrm{T}} J^*$ [18, Chapter 8]. In practice, only large perturbations give rise to indefinite Hessians.

[4] The distance from $\mathbf{x}$ to the surface has to be smaller that the radius of curvature of the surface near $\mathbf{x}$.

3.2.3 Summary of data set generation

I Given parameter values $\mathbf{u}^*$, generate points $X^* = \{\mathbf{x}_i^* : i \in I\}$ lying on the element $\mathcal{E}(\mathbf{u}^*)$. Calculate the outward unit normals $N^* = \{\mathbf{n}_i^* : i \in I\}$ to $\mathcal{E}(\mathbf{u}^*)$ at X^*.

II Calculate the Jacobian matrix J^* and (a basis for) the null space $Z^* = [\ldots \mathbf{z}_k \ldots]$ of $J^{*\mathrm{T}}$. Select or compute multipliers $\boldsymbol{\nu}$.

III Set

$$\left.\begin{array}{rcl} \boldsymbol{\delta} & = & \sum_k \nu_k \mathbf{z}_k, \\ X_{\boldsymbol{\delta}} & = & \{\mathbf{x}_i^* + \delta_i \mathbf{n}_i^*,\ i \in I\}. \end{array}\right\} \tag{10}$$

IV Calculate $H_{\boldsymbol{\delta}}$ and check if it is strictly positive definite; if not reduce the norm of $\boldsymbol{\delta}$ by reducing the multipliers e.g., $\nu_k = \nu_k/2$ and go to step III.

Step IV can be made more systematic, but as mentioned above it is not difficult to generate data sets which give rise to strictly positive definite Hessians.

Given X^*, N^* and Z^*, it is possible (by varying the multipliers $\boldsymbol{\nu}$) to generate many data sets $X_{\boldsymbol{\delta}}$ as in step III and we refer to the triple $\langle X^*, N^*, Z^* \rangle$ as a *reference base* for generating reference pairs $\langle X_{\boldsymbol{\delta}}, \boldsymbol{\delta} \rangle$.

Below is a module written in the Matlab 'language' [24] which generates a reference pair from a reference base and a set of multipliers $\boldsymbol{\nu}$. Note that this is a completely generic module in that it applies to all the geometric elements.

```
  function [Xd, d]   = rb2rp(Xs, Ns, Zs, nu, normd)
%
%  Generate data set Xd from the reference base < Xs, Ns, Zs >.
%
%  Input                Xs  - m x q array of points on element, q = 2 or 3.
%                       Ns  - m x q array of outward unit normals at Xs.
%                       Zs  - m x p array of vectors in the null space
%                             of the transpose of the Jacobian matrix
%                             associated with Xs.
%                       nu  - p x 1 array of multipliers.
%                    normd  - norm of the residual errors.
%
%  Output               Xd  - m x q array of data points.
%                        d  - m x 1 array of residual errors.
%                               d(i) = the distance from Xd(i) to
%                                      the element.
%                             Note: norm(d) = normd.
%
%
% Start calculations
% ----- ------------
%
% Check array lengths.
%
     [ mZ, p ] = size(Zs);
     [ mX, q ] = size(Xs);
%
     if p ~= length(nu) | mX ~= mZ
%
        error('Check array lengths!')
```

```
%
     end  % if
%
% Form residuals
%
   d = Zs*nu;
%
  nd = norm(d);
%
  if nd > eps*normd
     d = d*(normd/nd);
  end
%
% Form data points Xd.
%
   Xd = Xs + d(:,ones(1, q)).*Ns;
%
% ------------
% End of rb2rp
%
```

Example: Circle in the plane. The null space method can be illustrated for the case of the circle in the plane. Given centre coordinates (x_0, y_0) and radius r_0, and a set of angles $\Theta = \{\theta_i : i \in I\}$, we can generate a set of points

$$X^* = \{(x_i^*, y_i^*) = (x_0 + r_0 \cos\theta_i,\ y_0 + r_0 \sin\theta_i),\ i \in I\},$$

lying on the circle. The ith row of the Jacobian matrix J^* is $(\cos\theta_i, \sin\theta_i, 1)$ (see for example [14, section 6]). The following Matlab module forms a reference base given an array of angles θ_i.

```
  function [Xs, Ns, Js, Zs]   = g_cir2_rb(theta, x0, y0, r0)
%
% Generate a reference base < Xs, Ns, Zs > for the least-squares
% best-fit circle with centre (x0, y0) and radius r0.
%
%
%  Input             theta  - m x 1 array of angles.
%                             m > 3.
%                   x0, y0  - coordinates of the circle centre.
%                       r0  - radius.
%
%  Output               Xs  - m x 2 array of points on circle.
%                       Ns  - m x 2 array of outward unit normals
%                             to circle at Xs.
%                       Js  - m x 3 Jacobian matrix.
%                       Zs  - m x (m - 3) null space of Js'.
%
% Start calculations
% ----- ------------
%
     m = length(theta);
%
% Form Jacobian.
%
     c = cos(theta);
```

```
     s = sin(theta);
%
    Js = [ c  s  ones(m, 1) ];
%
% Find null space Zs.
%
    Zs = null(Js');
%
% Form data points xs = x0 + r0 cos theta_i, etc.
%
    Xs = [ (x0 + r0*c) (y0 + r0*s) ];
%
% Form normals.
%
    Ns = [ c s ];
%
% ----------------
% End of g_cir2_rb
%
```

This module can be used in conjunction with `rb2rp` to generate reference pairs. □

3.3 MINIMAL PERTURBATIONS AND FORM ERROR

In this section we consider the following problem:

Given a data set $X = \{\mathbf{x}_i : i \in I\}$ and a set of parameters $\mathbf{u}^*$ find the perturbed set $\hat{X}$ 'closest' to X for which $\mathbf{u}^*$ defines the best-fit element to $\hat{X}$.

If we consider only perturbations normal to $\mathcal{E}(\mathbf{u}^*)$, we can find a solution to this problem as follows. Determine $d_i = d(\mathbf{x}_i; \mathbf{u}^*)$ and the set of of unit outward normals $N^* = \{\mathbf{n}_i^* : i \in I\}$ such that $\mathbf{x}_i^* = \mathbf{x}_i - d_i\mathbf{n}_i^*$ lies on $\mathcal{E}(\mathbf{u}^*)$. Solve

$$\min_{\boldsymbol{\delta}} \|\boldsymbol{\delta} - \mathbf{d}\| \tag{11}$$

subject to the condition

$$J^{\mathrm{T}}\boldsymbol{\delta} = \mathbf{0}, \tag{12}$$

where J is the Jacobian matrix associated with X, and set

$$\begin{aligned} \hat{X} &= \{\hat{\mathbf{x}}_i : \hat{\mathbf{x}}_i = \mathbf{x}_i + (\delta_i - d_i)\mathbf{n}_i^*, \quad i \in I\}, \\ &= \{\hat{\mathbf{x}}_i : \hat{\mathbf{x}}_i = \mathbf{x}_i^* + \delta_i\mathbf{n}_i^*, \quad i \in I\}, \\ &= X_{\boldsymbol{\delta}}, \end{aligned}$$

in the previous notation.

Any $\boldsymbol{\delta}$ satisfying (12) can be written uniquely as a linear combination $\boldsymbol{\delta} = \sum_k \nu_k \mathbf{z}_k$ of basis vectors for the null space Z^* of J^{T} and so that the solution of (11) can be found by solving

$$\min_{\boldsymbol{\nu}} \|Z^*\boldsymbol{\nu} - \mathbf{d}\|. \tag{13}$$

We may wish to solve this problem for different norms. If we choose the Euclidean norm then this is a linear least-squares problem which can be accurately solved using an orthogonal factorisation approach [18, Chapter 6]. If we choose the Chebyshev norm so that (11) becomes

$$\min \max_i |\delta_i - d_i|$$

then (13) corresponds to a linear Chebyshev approximation problem which can be solved by linear programming methods [5]. If the best-fit element to X is determined by parameters close to $\mathbf{u}^*$ then $\hat{X}$ will be close to X and $\boldsymbol{\delta}$ will be close to $\mathbf{d}$.

To summarise, starting with a data set X, an element $\mathcal{E}$ defined by parameters $\mathbf{u}^*$ and corresponding distances $d_i = d(\mathbf{x}_i; \mathbf{u}^*)$, this procedure produces a reference base $\langle X^*, N^*, Z^* \rangle$ and a specific reference pair $\langle X_{\boldsymbol{\delta}}, \boldsymbol{\delta} \rangle$ which is in some sense closest to the original (non-reference) pair of X and $\mathbf{d}$. Furthermore, once we have one reference pair we can generate many others using null space perturbations as before.

One important use of this extra capability is to produce reference pairs which simulate known form deviations. The following example will illustrate the concept.

Example: three-lobed circle data set. Some manufacturing processes produce a lobing effect on surfaces of revolution. For this reason, we may wish to synthesise reference pairs which portray this type of form deviation. A three-lobed 'circle' can be described in polar coordinates by the equation

$$r(\theta) = r_0 + a\cos(3\theta + \theta_0),$$

If we choose a set of angles $\Theta = \{\theta_i : i \in I\}$ we can generate a data set

$$X = \{(x_i, y_i) : x_i = r(\theta_i)\cos\theta_i,\ y_i = r(\theta_i)\sin\theta_i,\ i \in I\}.$$

There is no guarantee that the best-fit circle to this data will be centred at $(0,0)$, but using the method outlined above we can find a perturbed set for which this is the case; see Figure 1.

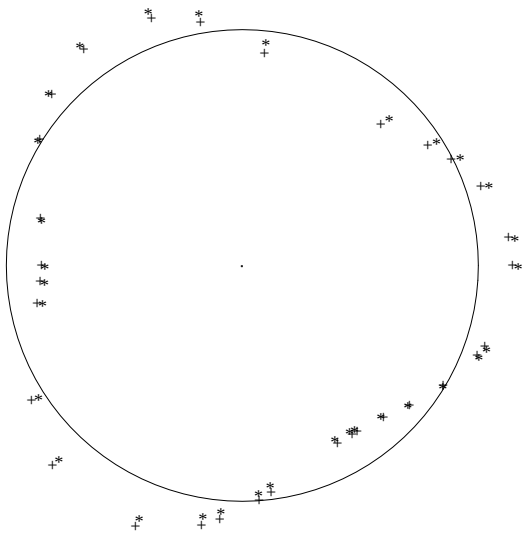

Figure 1. Points on three-lobed circles. Given the data points marked by $*$ lying on a three-lobed circle centred at the origin, we can find the closest set of points (marked by $+$) for which the best-fit circle is centred at the origin.

□

3.4 VALIDATION OF NULL SPACE METHODS

Although the generation of data sets using the null space methods outlined above is relatively straightforward, care has to be taken at various stages in order to obtain maximum accuracy.

Parametrization. The evaluation of the Jacobian matrix J^* depends on the choice of element parametrization $\mathbf{u} \mapsto \mathcal{E}(\mathbf{u})$. A poor parametrization may lead to an inaccurate estimate of J^* which, moreover, may be ill-conditioned giving rise to further error in the subsequent null-space calculations. Fortunately it is possible to find parametrizations of geometric elements which lead to good stability properties [2, 7]. Note, however, that from the mathametical point of view the null space Z^* is independent of parametrization: given a data set X and two parametrizations $\mathbf{u} \mapsto \mathcal{E}(\mathbf{u})$ and $\mathbf{v} \mapsto \mathcal{E}(\mathbf{v})$ with $\mathcal{E}(\mathbf{u}_0) = \mathcal{E}(\mathbf{v}_0)$, then the associated Jacobian matrices[5] J and K, say, evaluated at $\mathbf{u}_0$ and $\mathbf{v}_0$, respectively, are related by a non-singular matrix V such that $K = JV$, and

$$K^{\mathrm{T}}\mathbf{q} = 0 \iff V^{\mathrm{T}}J^{\mathrm{T}}\mathbf{q} = 0 \iff J^{\mathrm{T}}\mathbf{q} = 0.$$

Formulæ for distances and their derivatives. It is imperative that the formula for the distance from a point to an element in terms of its parameters are algebraically correct and evaluated in a numerically stable manner. Such formulæ are given in [14, 7, 2] for a number of parametrizations. It is also essential that the associated Jacobian matrix is calculated accurately. Three computational tools can be used to verify that all expressions are correct: i) symbolic algebra packages, e.g. [21], ii) differentiation arithmetic packages [19] and iii) finite difference approximations.

Null space calculations. Having determined an accurate Jacobian matrix J^*, it is necessary to find a basis for the null space Z^* of $J^{*\mathrm{T}}$. QR factorisation algorithms using orthogonal transformations, with their excellent stability properties [18], can be employed to perform this task.

3.5 GENERALISATIONS

General curves and surfaces. The null space method for generating reference bases for testing least-squares element fitting software applies to any orthogonal distance regression problem (see e.g. [6, 9]).

Other approximation problems. The null space method is based on sufficiency conditions for a local minimum to the element-fitting problem. The solutions of other approximation problems can be analysed in the same way to give corresponding sufficiency conditions from which a data set generation strategy can be developed. In particular, it is possible to generate data sets for which the Chebyshev best-fit element is known; see for example [13, Chapter 9], [17, Chapter 3] for discussions on optimality conditions. Algorithms for Chebyshev element fitting are described in [4, 20].

[5] We assume that both J and K are of full rank.

4 INVARIANCE TESTING

We now wish to consider methods of testing element-fitting software which do not rely on reference results (whether or not these are generated by null space methods or through the application of reference software) but instead exploit various mathematical properties of least-squares fits. For example, if the points in a data set X are translated, rotated or scaled by a transformation T to form a new data set $T(X)$ then the best-fit element undergoes the same transformation (in exact arithmetic); in notation:

$$\mathcal{E}^*_{T(X)} = T(\mathcal{E}^*_X).$$

By comparing the computed solutions for X and $T(X)$ produced by software, it is possible to learn much about the stability of the underlying algorithms. A particularly simple but potentially revealing transformation to perform on a data set is to permute the order of the x-, y-, and z-components (corresponding to reflections and 90^o rotations), an operation which introduces no rounding error. Another simple observation which can be exploited in this manner is that the best-fit element is independent of the ordering of the points in the data set. Examples of how these ideas can be used on practice are given in section 9.

5 COMPARING BEST-FIT ELEMENT RESULTS

In this section we consider the second question posed in the introduction: how to compare the results from one algorithm with those from another.

5.1 COMPARISON BASED ON PARAMETER VALUES

A determination of the 'closeness' of two geometric elements $\mathcal{E}_1$ and $\mathcal{E}_2$ can be made by comparing parameter values for a parametrization $\mathbf{u} \longrightarrow \mathcal{E}(\mathbf{u})$ which can specify both instances. There will be parameter points $\mathbf{u}_1$ and $\mathbf{u}_2$ such that $\mathcal{E}_j = \mathcal{E}(\mathbf{u}_j)$, $j = 1, 2$, and 'closeness' can be quantified by $\|\mathbf{u}_1 - \mathbf{u}_2\|$ for some specified norm $\| \ \|$.

This method of comparison depends on the existence of a common specification of both elements. However, there are many ways of specifying geometric elements and it is quite likely that an element-fitting algorithm will employ a different specification from that specifying the reference solution. For example, to describe the location of a cylinder axis, some give the intersection of the axis with a fixed plane while others give the point on the axis nearest the centroid of the data. Thus, a test of software for conformance based on parameter values would seem to require either i) the testing body transforms the parameters output by the software under test to the reference parametrization, or ii) the testing body specifies *a priori* which parametrization should be used to specify the solution element. In the former situation, the testing body will have to design bespoke software to perform the parameter transformation, while in the latter, this (non-trivial) burden is placed on each enterprise requiring certification. In both cases, the testing of the element-fitting software is clouded by the presence of extraneous software.

On top of these difficulties lies a more fundamental question as to whether the difference in parameters values always gives a meaningful indication of the closeness of solutions. For example, in fitting circles to data lying on a small arc two algorithms may generate best-fit arcs which are close to each other but whose centres are significantly distant from each other. The underlying problem is that, in general, no one parametrization is suitable to specify all instances of a geometric element [2]. Although some parametrizations are more 'natural' than

others, there is no intrinsic or 'characteristic' choice of parametrization which can be used to quantify differences in solutions.

5.2 COMPARISON BASED ON RESIDUAL ERRORS

In contrast to the problems posed by the use of parameter values, the vector of residual errors for the best-fit element of a set of data is intrinsic to the data fitting problem. The residual errors are independent of parametrization and other algorithmic choices such as coordinate system and moreover satisfy mathematical invariance properties with respect to translation, rotation, scaling and order. Two solutions are the same only if their residual error vectors are equal.

There are a number of other reasons why the use of residual errors is particularly appropriate. Firstly, residual errors give meaningful information about the position of the element: small changes in the position of the element near the data are reflected in small changes in the residual values and *vice versa*. This means that differences in residual errors between two solutions accurately reflect the differences in the position of the element's curve or surface near the data. Secondly, null space methods automatically generate reference values for the residual errors. Thirdly, many calculations such as circularity, cylindricity, etc., in coordinate metrology involve only the residual errors, and do not refer to parameter values. Lastly, if parameter values are required for some reason they can be reconstructed from the residual errors: see section 6.

To be able to compare residual errors, element-fitting software must output estimates of the distances from points to the computed best-fit element. Since the software aims to minimise such distances, this information should in principle be readily available. It is also is a recommendation in current Standards [7] for element fitting software, and therefore conformance testing based on residual error comparison should impose no extra burden on software designers. Some algorithms may for convenience compute only approximate distances, and it is desirable that a software evaluation detect such approximations (see below).

6 RESIDUAL ERRORS, REFERENCE SOFTWARE AND PARAMETER DETERMINATION

Suppose that in fitting an element $\mathcal{E}$ to a data set $X = \{\mathbf{x}_i : i \in I\}$, software S outputs residual errors $\hat{\mathbf{d}}$ at the computed solution, and that reference software employing a parametrization $\mathbf{u} \mapsto \mathcal{E}(\mathbf{u})$ computes the minimum $\mathbf{u}^*$ of the error function

$$D(X, \mathbf{u}) = \sum_{i \in I} d^2(\mathbf{x}_i; \mathbf{u}).$$

Consider the related function

$$F(\mathbf{u}) = F(X, \hat{\mathbf{d}}; \mathbf{u}) = \sum_{i \in I} (d(\mathbf{x}_i; \mathbf{u}) - \hat{d}_i)^2 \tag{14}$$

which measures, in terms of the parameters $\mathbf{u}$, the difference between the computed residuals $\hat{\mathbf{d}}$ and the distances $d(\mathbf{x}_i; \mathbf{u})$. We observe that $F = 0$ *if and only if* there exists $\hat{\mathbf{u}}^*$ such that $\hat{d}_i = d(\mathbf{x}_i, \hat{\mathbf{u}}^*)$, $\forall i \in I$ and, furthermore, this $\hat{\mathbf{u}}^*$ (if it exists) can be found by minimising $F(X, \hat{\mathbf{d}}, \mathbf{u})$ with respect to $\mathbf{u}$. We can use this fact in the following manner. Given $\hat{\mathbf{d}}$ we

determine the minimiser $\hat{\mathbf{u}}^*$ of F. If the value of F at the minimum is zero[6], then we know from above that the distances $\hat{\mathbf{d}}$ calculated by S are the distances to the element $\mathcal{E}(\hat{\mathbf{u}}^*)$ and it can be inferred that the software S has found the best-fit element specified by $\hat{\mathbf{u}}^*$. Thus, without knowledge of the parametrization employed by $\mathbf{S}$, the computed solution can be expressed in terms of the parametrization used by the reference software and is thus available for comparison with the reference solution $\mathbf{u}^*$. If, on the other hand, the value of F at $\hat{\mathbf{u}}^*$ is significantly different from zero this indicates that the software S calculates only approximate distances.

It is noted that reference element-fitting software can easily be adapted to provide software to minimise $F(X, \hat{\mathbf{d}}; \mathbf{u})$.

7 DESIGN OF TEST DATA SETS

In this section, we discuss the desirable properties a suite of data sets should have in order that they will lead to a thorough test of element-fitting software.

Data sets for testing software should of course be designed to check, where appropriate, the correctness of software under test. More importantly, they should also expose weaknesses in the software [25, 10, 3, 8]. To accomplish this, the data must be as varied as possible and where possible test the software to its limits. Data sets should therefore be designed to test the software under 'good' conditions, corresponding to accurate data representative of the element in question, and under 'bad' conditions, corresponding to inaccurate or unrepresentative data. Thus, a test suite should ideally incorporate design features to test the following attributes of assessment software:

independence with respect to transformations (rotations and translations) of the data;

independence with respect to data ordering;

reliability of the results for varying distributions of points;

reliability of the results for data representing widely differing forms of a particular geometry (e.g. cones with a very small or very large apex angle);

reliability of the results for data with widely differing magnitudes of form error and random error;

behaviour on a small (minimal or near minimal) number of points;

behaviour on a large number of data points, for example with respect to computation time and memory requirements.

To test all these attributes thoroughly for each geometric element would require a very large suite. A more modest sized suite will necessarily be less comprehensive than the ideal but in designing such a suite it is fruitful to bear these features in mind.

We now consider how well the null space method can match these design aims. The null space method is in fact an extremely flexible tool for generating data sets. We have seen that given a reference base each choice of multipliers $\boldsymbol{\nu}$ determines a test data set. By using a random

[6]The value of F should be compared with $\epsilon^2\|\hat{\mathbf{d}}\|^2$ where ϵ is the machine precision.

number generator to generate these coefficients, it is possible to provide automatically any number of reference pairs.

The test set designer also has the freedom to specify the distribution of the points X^* lying on the element. For example, random scatters of points or uniform regular distributions can be chosen. More importantly, the degree of 'representativeness' can be varied so as to give the software under test a range of problem difficulty. For example, it is known [15] that standard circle-fitting algorithms employing a centre and radius parametrization can perform poorly if the data covers only a small arc of the circle. We can generate circle data with a distribution of points limited to 120^o, 30^o, 5^o or 1^o arcs. Most importantly, in calculating the corresponding reference base, the associated Jacobian matrices can be evaluated using a parametrization stable for small arcs [15] in order to ensure that the reference pair is fully accurate. The flexibility in the null space approach allows the test data designer to use the most appropriate parametrization for each data set. By contrast, the software under test will usually employ only one parametrization [7].

8 PRACTICAL CONFORMANCE TESTING

8.1 BASIC PROCEDURE

Conformance testing of least-squares element-fitting software can be made very straightforward. We start with a suite of data sets X_q, $q = 1, \ldots, n_Q$, and for each data set X_q a set of reference residual errors $\mathbf{d}_q$. These can be distributed by any standard computer readable medium in standard code, for example, ASCII on 5.25in floppy disk. The software under test calculates for each data set X_q a set of residual errors $\hat{\mathbf{d}}_q$ and these results can be again stored and distributed in standard form. The errors $\mathbf{d}_q$ and $\hat{\mathbf{d}}_q$ can then be compared by looking at $\|\mathbf{d}_q - \hat{\mathbf{d}}_q\|$ for some norm $\| \ \|$. The simplicity of the procedure is apparent and it forms the basis of a proposed Standard for conformance testing of form assessment software [1]. By contrast, implementing conformance testing procedures based on comparison of parameter values requires considerable effort; see for example [27, 11] which report on intercomparison exercises for element-fitting software.

A second level of analysis can be introduced relatively painlessly by exploiting the ideas of section 6. Given a set of residuals $\hat{\mathbf{d}}_q$, an estimate of the corresponding fitted parameters $\hat{\mathbf{u}}$ can be obtained by minimising $F(X_q, \hat{\mathbf{d}}, \mathbf{u})$ given by (14). The value of F at the minimum also will be of interest.

8.2 COMPUTER INTENSIVE SOFTWARE TESTING.

Given a reference base $\langle X^*, N^*, Z^* \rangle$, each choice of multipliers $\boldsymbol{\nu}$ yields a reference pair $\langle X_q, \mathbf{d}_q \rangle$. By using a random number generator to generate these multipliers, software S can be tested on thousands of data sets as follows:

For $q = 1$ **to** n_Q**:**

I Generate $\boldsymbol{\nu}$ and determine $\langle X_q, \mathbf{d}_q \rangle$ as in (10).

[7] One major feature of the reference software package LSGE [26] is that the algorithms, when appropriate, use a *family* of parametrizations [2, 14] for each element.

II Run S on X_q to calculate residual errors $\hat{\mathbf{d}}_q$.

III Record $\Delta_q = \|\mathbf{d}_q - \hat{\mathbf{d}}_q\|$ and $\rho_q = \Delta_q / \|\mathbf{d}_q\|$.

IV Calculate $\hat{\mathbf{u}}_q = \min_{\mathbf{u}} F(X_q, \hat{\mathbf{d}}_q, \mathbf{u})$ and record $\hat{\mathbf{u}}_q$, $F_q = F(X_q, \hat{\mathbf{d}}_q, \hat{\mathbf{u}})$ and $\phi_q = F_q / \|\mathbf{d}_q\|$.

Next q**;**

Any systematic bias can be detected from the information gathered at step IV. More elaborate testing regimes are also possible in which the software is tested for a range of data distributions and element position and orientations.

Example: circle data sets. The following Matlab module `g_cir2_th` generates at random different distributions of points (specified by angles) around a circle. This module can be used with `g_cir2_rb` and `rb2rp` to generate reference pairs for circles with random distribution of points, centre coordinates and radii.

```
 function [ theta ] = g_cir2_th(m)
%
% Generate a random set of angles.
%
%   Input
%                m  - number of angles.
%
%  Output
%            theta  - m x 1 array of angles.
%
% Start calculations.
% -------------------
%
   rand('uniform')
%
% Select a length of arc and initial angle.
%
        l = 2*pi*rand(1,1);
%
   theta0 = 2*pi*rand(1,1);
%
   theta = theta0 + l*rand(m,1);
%
% End of g_cir2_th.
% -----------------
%
```

□

9 CASE STUDY: CIRCLE ALGORITHMS

We illustrate some of the ideas described in the sections above for the case of finding the least-squares best-fit circle to data in the xy-plane.

We test five algorithms A, B, C, D and E on four data sets X_1, X_2, X_3 and X_4 pictured in Figure 2 and listed in Appendix 1. The data sets were generated using the null space method

so that the best-fit circle to each data sets is centred at the origin and has radius 100. The first three data sets correspond to randomly placed points along arcs subtending angles of 360^o, 120^o and 45^o, respectively. The fourth data set represents the form deviation of a three-lobed circle and was generated using the minimal perturbation technique described in section 3.3. To each data set X_k we also have a set $\mathbf{d}_k$ of reference residuals errors (also listed in Appendix 1) with $\|\mathbf{d}_k\| = 1$, $k = 1, 2, 3, 4$.

The algorithms have been implemented in Matlab. The tests were performed on a DEC VAX running VMS with machine precision $\epsilon = 1.3878e - 17$.

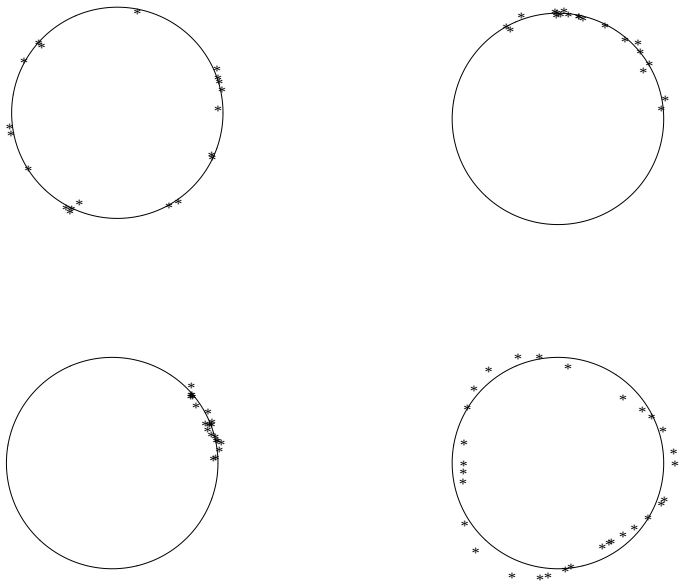

Figure 2. Four data sets X_k, $k = 1, 2, 3, 4$, of points lying close to a circle. Three data sets correspond to random error while the fourth (lower right) synthesizes a lobing form error. The residual errors have been exaggerated.

9.1 TEST RESULTS ON DATA SETS X_1, X_2, X_3 AND X_4

Table 1 shows the difference between the centre coordinates determined by the five algorithms and the reference values of $(0, 0)$. Only Algorithm A gives excellent agreement. Algorithms B, D and E give approximately the same values for the centre coordinates while algorithm C gives different values but comparable in error. Note that for data set X_3, the error in the centre coordinates corresponding to algorithms B, C, D and E is of the order of 0.1 which is quite large considering that the root-mean-square error is approximately 0.05.

The differences between the radii of the best-fit circles and the reference value of 100 are given in Table 2. These results are entirely consistent with the results of Table 1. Note that algorithms B, D and E underestimate the radii for data sets X_2 and X_3 while algorithm C overestimates them.

Table 1. Centre coordinates of the best-fit circles determined by the five algorithms A, B, C, D and E on the four data sets X_1, X_2, X_3 and X_4. These values should be compared with the reference values of $(0, 0)$.

	A	B	C	D	E
X1	1.1102e-16	-4.4251e-05	-1.4901e-04	-4.4251e-05	-4.4251e-05
	4.4409e-16	-6.0049e-04	-2.0743e-04	-6.0049e-04	-6.0049e-04
X2	0	8.3359e-03	-8.6282e-03	8.3359e-03	8.3359e-03
	-1.7764e-15	1.5442e-02	-1.8370e-02	1.5442e-02	1.5442e-02
X3	-8.8818e-15	8.7815e-01	-9.0407e-01	8.7815e-01	8.7815e-01
	-2.6645e-15	3.9484e-01	-4.0698e-01	3.9484e-01	3.9484e-01
X4	0	-9.1469e-04	-2.0803e-03	-9.1469e-04	-9.1469e-04
	-2.2204e-16	-2.6568e-03	-1.0497e-04	-2.6568e-03	-2.6568e-03

Table 2. Difference between the radii of the best-fit circle determined by the five algorithms A, B, C, D and E on the four data sets X_1, X_2, X_3 and X_4 and the reference value of 100.

	A	B	C	D	E
X1	0	1.6636e-04	1.2249e-03	1.6636e-04	1.6636e-04
X2	0	-1.4454e-02	1.8370e-02	-1.4454e-02	-1.4454e-02
X3	8.8818e-15	-9.3738e-01	9.6723e-01	-9.3738e-01	-9.3738e-01
X4	0	1.8894e-03	1.1077e-02	1.8894e-03	1.8894e-03

Table 3 compares the norm of the difference between the residual errors calculated by the five algorithms with the reference residuals. Again only algorithm A gives good agreement. Note that algorithm E gives different results from algorithms B and D. This is at variance with the results displayed in Tables 1 and 2.

Table 3. The Euclidean norm $\|\hat{\mathbf{d}}_k - \mathbf{d}_k\|$ of the difference between the residual errors $\hat{\mathbf{d}}_k$ calculated by the five algorithms A, B, C, D and E on the four data sets X_1, X_2, X_3 and X_4 and the reference residual errors $\mathbf{d}_k$, $k = 1, 2, 3, 4$.

	A	B	C	D	E
X1	4.5996e-15	2.0017e-03	5.6353e-03	2.0017e-03	2.2687e-03
X2	7.8732e-15	1.1774e-02	1.4724e-02	1.1774e-02	1.2104e-02
X3	4.6419e-15	9.7122e-02	9.8286e-02	9.7122e-02	9.7242e-02
X4	6.4921e-15	1.5874e-02	6.1332e-02	1.5874e-02	1.2512e-02

9.2 TRANSLATION INVARIANCE TESTING

In section 4, we described how invariance properties of the solution can be used to test software against itself.

We illustrate how this can be done by comparing the residual errors $\hat{\mathbf{d}}$ calculated for a data set $X = \{(x_i, y_i) : i \in I\}$ with the residual errors $\hat{\mathbf{d}}_T$ calculated for a translated set $T(X) = \{x_i + x_0, y_i + y_0\}$. Mathematically, these two sets of residual errors should be equal but due

to rounding error they will be different. How different will depend on the numerical stability and the invariance properties enjoyed by the algorithm implemented. An important point is that while the *stated* computational aim of software may have an invariant property, the *actual* computational aim of the algorithm implemented might not.

Table 4 gives the norms of the difference of the computed residual error vectors for data sets X_2 and six translations of X_2 with translation constants $x_0 = y_0 = 10^p$, $p = 1, \ldots, 6$. Note that these comparisons involve only the computed residual errors and do not refer to the reference residual errors. The results for algorithms A and B show that these differences are comparable with the loss of precision incurred in forming the translated sets. Algorithm C is seen to be sensitive to translation while algorithms D and particularly E exhibit numerical instability.

Table 4. The Euclidean norm $\|\hat{\mathbf{d}}_T - \hat{\mathbf{d}}\|$ of the difference between the residual errors $\hat{\mathbf{d}}_T$ calculated by the five algorithms A, B, C, D and E on the four data sets X_1, X_2, X_3 and X_4 translated by $(10^p, 10^p)$, $p = 1, \ldots, 6$, and the residual errors $\hat{\mathbf{d}}$ calculated for the original data sets.

	A	B	C	D	E
10^1	0	6.4047e-15	4.6544e-03	1.0950e-14	7.0111e-15
10^2	1.1916e-14	6.6465e-15	9.1300e-02	1.0204e-14	3.4058e-13
10^3	3.8918e-14	3.9201e-14	2.7745e-02	2.1823e-12	3.2977e-10
10^4	2.8758e-13	2.8994e-13	2.5821e-02	4.3403e-11	1.1933e-07
10^5	1.9592e-12	1.9616e-12	2.5641e-02	3.4682e-08	2.7444e-04
10^6	1.8351e-11	1.8352e-11	2.5623e-02	2.8082e-06	4.1007e-01

9.3 RECOVERY OF PARAMETER VALUES FROM RESIDUAL ERRORS

Section 6 showed how fitted parameter values could be estimated from the computed residual errors by minimising the function $F(X, \hat{\mathbf{d}}, \mathbf{u})$ given by (14). Applying this to these results for data set X_1, we can compare the actual centre coordinates calculated by algorithms B, C, D and E with the ones estimated by minimising the appropriate F. The difference in these coordinates is given in the first column of Table 5 and shows that for algorithm B, C and D,

Table 5. The first (numerical) column gives the difference between the centre coordinates calculated by the four algorithms B, C, D and E and those calculated by minimising the function $F(X, \hat{\mathbf{d}}, \mathbf{u})$ for the data set X_1. The second column gives the square root of the minimum of F for the four data sets.

	centres	sqrt(F)
B	0	0
	0	
C	6.9860e-16	2.5121e-15
	-1.9947e-16	
D	7.8957e-17	0
	-3.3457e-16	
E	-3.2264e-05	1.8759e-03
	-1.3502e-04	

the method successfully determines the actual computed parameters. Its failure to do so for algorithm E is explained by the second column which gives the minima of F obtained for the four algorithms. The fact that the minimum for E is significantly different from zero indicates that algorithm E calculates only approximate residual errors.

9.4 DISCUSSION

The limited testing of the five algorithms carried out above has produced a considerable amount of information and it is interesting to view the results in light of the algorithm descriptions which we now give.

Algorithm A uses the Gauss-Newton algorithm to minimise the (nonlinear) sum of squares D defined by (4) and (2) (see [14, Section 6]) and is a module in NPL's reference software package LSGE [26]. Algorithms B, C, D and E minimise an error function of the form

$$\min_{\mathbf{u}} \sum_{i \in I} (u_1(x_i^2 + y_i^2) + u_2 x_i + u_3 y_i + u_4)^2,$$

with $u_1 = 1$ in the case of algorithms B, D and E and $u_3 = 1$ in the case of algorithm C (see [15]). These latter four formulations give rise to linear least-squares optimisation problems which can be solved using an orthogonal factorisation approach or by solving the associated normal equations. Algorithms A and B also perform an initial translation of the data so that the mean of the transformed data points lies at the origin. Algorithm E estimates the distance of point $\mathbf{x}$ to a circle with centre (a, b) and radius r_0 by

$$\frac{r^2 - r_0^2}{2r_0} = (r - r_0)\frac{r + r_0}{2r_0}.$$

where $r = [(x-a)^2 + (y-b)^2]^{1/2}$. This information is summarised in Table 6.

Table 6. Summary of characteristics of the five algorithms under test. OF = orthogonal factorisation, NE = normal equations, CR = correct residuals, AR = approximate residuals.

A	Gauss Newton	OF	centering	CR
B	linear least squares $u_1 = 1$	OF	centering	CR
C	linear least squares $u_3 = 1$	OF	no centering	CR
D	linear least squares $u_1 = 1$	NE	centering	CR
E	linear least squares $u_1 = 1$	NE	no centering	AR

The results reflect the good numerical properties which result from centering and the use of orthogonal factorisations. The lack of invariance in the constraint $u_3 = 1$ (algorithm C) is shown up in Table 4. The presence of approximate residuals is detected in Table 5.

It would be reassuring to think that the poorly-performing algorithms C and E are a far cry from the algorithms that are implemented as assessment software in coordinate measuring machines but in fact the use of linear approximations and normal equations is widespread; see, for example, the reports of intercomparisons of assessment software [11, 27]. Incidentally, algorithm C is a useful module in fitting circles to small arcs of data [15].

10 SUMMARY AND CONCLUSIONS

Conformance testing of assessment software is an important step in improving the quality of software used in coordinate metrology. We have described methods for generating test data and reference results – reference pairs – which can be used to test least-squares element-fitting software. These methods open the way for well-founded, practical conformance testing procedures.

11 ACKNOWLEDGEMENTS

The work reported on here was carried out as part of the 'Mathematical Algorithms for Metrology' project in the National Physical Laboratory's Length programme. This programme is in support of the National Measurement System and is funded by the Department of Trade and Industry. Project members G T Anthony, B P Butler, S A Hannaby and Dr P M Harris have contributed to this work. We thank our colleagues Susan M Hodson for her valuable comments on a draft on this report and Roger S Scowen for helpful discussions.

12 REFERENCES

[1] G. T. Anthony. Proposed draft British Standard: Method for testing CMM geometric form assessment software. Submitted to ISO/TC3/WG10 for discussion., 1991.

[2] G. T. Anthony, H. M. Anthony, M. G. Cox, and A. B. Forbes. The parametrization of fundamental geometric form. Technical Report EUR 13517 EN, Commission of the European Communities (BCR Information), Luxembourg, 1991.

[3] G. T. Anthony and M. G. Cox. Design and validation of software for dimensional metrology. Technical Report DITC 50/84, National Physical Laboratory, Teddington, 1984.

[4] G. T. Anthony *et. al.* Chebyshev reference software for the evaluation of CMM data. Technical report, Commission of the European Communities (BCR Information), Luxembourg. *In preparation.*

[5] I. Barrodale and C. Phillips. Solution of an over determined system of linear equations in the Chebyshev norm. *ACM Trans. Math. Softw.*, 1(3):264–270, September 1975.

[6] P. T. Boggs, R. H. Byrd, and R. B. Schnabel. A stable and efficient algorithm for nonlinear orthogonal distance regression. *SIAM Journal of Scientific and Statistical Computing*, 8(6):1052–1078, 1987.

[7] British Standards Institution, London. *BS 7172: British Standard Guide to the Method of Assessment of Position, Size, and Departure from Nominal Form of Geometric Features*, 1989.

[8] M. G. Cox. Improving CMM software quality. Technical Report DITC 194/92, National Physical Laboratory, Teddington, January 1992.

[9] M. G. Cox and A. B. Forbes. The reconstruction of workpiece surfaces from probe coordinate data. In *Proceedings of the Mathematics of Surfaces V Conference. In preparation.*

[10] M. G. Cox and K. Jackson. Algorithms and software for engineering metrology: a statement of need. Technical Report MOM 65, National Physical Laboratory, Teddington, 1983.

[11] R. Drieschner, B. Bittner, R. Elligsen, and F. Wäldele. Testing coordinate measuring machine algorithms phase II. Technical Report EUR 13417 EN, Commission of the European Communities (BCR Information), Luxembourg, 1991.

[12] R. Fletcher. *Practical Optimization, Vol. 1.* John Wiley and Sons, Chichester, 1980.

[13] R. Fletcher. *Practical Optimization, Vol. 2.* John Wiley and Sons, Chichester, 1981.

[14] A. B. Forbes. Least-squares best-fit geometric elements. Technical Report DITC 140/89, National Physical Laboratory, Teddington, 1989.

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

[16] A. B. Forbes. Geometric tolerance assessment. Technical Report DITC 210/92, National Physical Laboratory, Teddington, October 1992.

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

[18] G. H. Golub and C. F. Van Loan. *Matrix Computations.* North Oxford Academic, Oxford, 1983.

[19] A. Griewank and G. F. Corliss, editors. *Automatic Differentiation of Algorithms: Theory, Implementation and Applications*, Philadelphia, 1991. Society for Industrial and Applied Mathematics.

[20] P. M. Harris *et. al.* Chebyshev best-fit geometric elements by mathematical programming. Technical report, National Physical Laboratory, Teddington. *In preparation.*

[21] A. C. Hearn. *REDUCE user's manual.* Rand Corporation, Stanta Monica, CA, 1987.

[22] ISO/IEC, Geneva. *ISO/IEC 9646-1, Information technology — Open systems interconnection — Conformance testing methodology and framework — Part 1: General concepts*, 1991.

[23] C. B. Jones. *Software development: A rigorous approach.* Prentice Hall International, Englewoods Cliffs, NJ, 1980.

[24] C. Moler, J. Little, and S. Bangert. *Pro-Matlab for VAX/VMS Computers.* South Natwick, Mass., 1988.

[25] G. J. Myers. *The Art of Software Testing.* Wiley, New York, 1979.

[26] NPL. LSGE: Package of algorithms for least-squares geometric elements. National Physical Laboratory, 1991. Document ITCh 069.

[27] C. Porta and F. Wäldele. Testing of three coordinate measuring machine evaluation algorithms. Technical Report BCR EUR 10909 EN, Commision of the European Communities, Luxembourg, 1986.

[28] J. H. Wilkinson and C. Reinsch. *Handbook of Automatic Computation Volume II: Linear Algebra.* Springer-Verlag, Berlin, 1971.

APPENDIX 1 Four reference pairs for circle fitting.

In this appendix we list the x- and y-coordinates of the data points in the four data sets X_1, $k = 1, 2, 3, 4$ used in the case study (section 9) along with the corresponding reference residual errors $\mathbf{d}_k$ (in the third column).

$\langle X_1, \mathbf{d}_1 \rangle$

```
 1.93549821172177e+01   9.79769093068946e+01  -1.29633573871044e-01
 9.57528923715935e+01   2.91577954272579e+01   9.39230508215555e-02
-4.33356847217196e+01  -9.04133093118751e+01   2.62396096567960e-01
-4.29884560948020e+01  -9.03165712036788e+01   2.54487188330082e-02
 9.15345224486746e+01  -3.98203998677955e+01  -1.78994968303186e-01
-7.41497962918795e+01   6.66225575320223e+01  -3.16714228337892e-01
-9.94497519199023e+01  -1.21930887133134e+01   1.94433823944775e-01
 4.87315570437628e+01  -8.73826849675412e+01   5.24776507112007e-02
 9.77860516034130e+01   2.15819031539315e+01   1.39355060490767e-01
 9.45018584784810e+01   3.29944735214684e+01   9.61364831116955e-02
-9.84314413289023e+01  -1.85847038210405e+01   1.70553847929484e-01
-4.76108778017298e+01  -8.81051105833550e+01   1.46423780165259e-01
 9.95188466141078e+01   4.81739754863515e+00  -3.64624000589427e-01
-7.43505941282206e+01   6.68759760099370e+01   2.03505179810227e-03
 9.15764739467267e+01   4.08935801462314e+01   2.92250337160959e-01
-8.69549595730067e+01   4.96083897868236e+01   1.10725357399386e-01
-3.84488695434761e+01  -9.16371167565909e+01  -6.23560455043900e-01
-8.47042346683035e+01  -5.29940423857317e+01  -8.41559154717372e-02
 9.05590036961778e+01  -4.23108619982937e+01  -4.42988445231115e-02
 5.69017475744348e+01  -8.24219627838287e+01   1.55822727206146e-01
```

$\langle X_2, \mathbf{d}_2 \rangle$

```
 4.50162818869036e+01   8.92387847647143e+01  -4.98807344174504e-02
 9.84465228699529e+01   1.91998862180572e+01   3.01313530633792e-01
 1.99683606494525e+01   9.78876613209570e+01  -9.63981334357767e-02
 6.43146401245176e+01   7.64300140890310e+01  -1.10461068259547e-01
 1.02008068363258e+01   9.94387993244683e+01  -3.93513866386228e-02
-3.30604855400386e+01   9.47317765049505e+01   3.34964912173043e-01
-2.55861148869710e+00   1.00129302566103e+02   1.61987425990308e-01
 8.53686206566653e+01   5.23023980828201e+01   1.16643162040792e-01
 9.93754547715486e+01   9.91152970467919e+00  -1.31489286468092e-01
 2.91280310593691e+00   9.98962181840977e+01  -6.13246564734123e-02
 7.73421922912906e+01   6.35027118985898e+01   7.20197003052144e-02
 2.42931334024033e+01   9.68339500066733e+01  -1.65285574596489e-01
-1.34082573443752e+00   9.98611754989661e+01  -1.29823342073563e-01
-4.83835075797298e+01   8.75581102718304e+01   3.69255829654708e-02
 7.24240132142278e+01   6.95508114996576e+01   4.11916973612010e-01
 8.64330421493972e+01   4.91899669373060e+01  -5.49893803512039e-01
-4.66447194307465e+01   8.80741131252224e+01  -3.36670467179735e-01
 5.73632992173228e+00   1.00071027806156e+02   2.35303596843894e-01
-7.03523470864651e-01   1.00094964242082e+02   9.74365900430903e-02
 2.04599083938669e+01   9.77845342325703e+01  -9.79330215528872e-02
```

$\langle X_3, \mathbf{d}_3 \rangle$

```
  9.96891292996286e+01   5.69717942747770e+00  -1.48208058408193e-01
  8.81697562716082e+01   4.77200791809109e+01   2.55233669007023e-01
  7.67860713968863e+01   6.40025861953083e+01  -3.78481611385748e-02
  9.77997836390940e+01   2.12736484806364e+01   8.67913339781028e-02
  9.38971942130368e+01   3.35064228838823e+01  -3.03643719771246e-01
  8.21947438863170e+01   5.64745460678640e+01  -2.73622967097930e-01
  9.31456174493418e+01   3.66871460906623e+01   1.10202968012847e-01
  9.86738268410277e+01   1.86443447242552e+01   4.19797319303386e-01
  9.78659654835885e+01   2.14649608099385e+01   1.92273866836330e-01
  9.58765066570363e+01   2.77946905152846e+01  -1.75907970083863e-01
  9.93417719548240e+01   1.30659349409542e+01   1.97336845873841e-01
  9.27278032885847e+01   3.72637094981778e+01  -6.48733012708647e-02
  7.60215951152839e+01   6.46869417612409e+01  -1.81748370879912e-01
  7.54976124112452e+01   6.54312210657152e+01  -9.43736822139865e-02
  9.95738974467773e+01   4.74001820328752e+00  -3.13346804555493e-01
  7.57872644517343e+01   6.52089420559330e+01  -2.04242000840549e-02
  9.24284367118346e+01   3.86703442510910e+01   1.91873111003890e-01
  9.16020071943372e+01   3.93233985428742e+01  -3.14206654141939e-01
  9.69559912197837e+01   2.48155874204510e+01   8.13549690091400e-02
  7.17296531160392e+01   7.02401561884686e+01   3.93339806621495e-01
```

$\langle X_4, \mathbf{d}_4 \rangle$

```
  8.38562691821900e+01   5.37600680958143e+01  -3.90669097606640e-01
  9.51493699226887e+01  -3.24005126970891e+01   5.14654751030644e-01
  8.94065693698522e+01   4.46732812738261e+01  -5.38309575770686e-02
 -9.91041338903626e+01  -4.40303496224103e-01  -8.94888016168885e-01
 -7.50089943756386e+01   6.68513076072833e+01   4.76099476682509e-01
 -1.69743686451250e+01   9.87907081128944e+01   2.38381872153628e-01
  8.56666971228509e+01  -5.15261000439241e+01  -3.13950198626307e-02
 -9.75740075949552e+01  -1.84527835941789e+01  -6.96464410807706e-01
 -9.67929375864453e+01   2.19899329612640e+01  -7.40592797184291e-01
  9.35903103721402e+01  -3.63900472188946e+01   4.16043201009066e-01
  9.54529529191280e+01   3.10702407438932e+01   3.82399258359793e-01
  7.23006971468979e+00  -9.97626276437781e+01   2.42759657394214e-02
  1.26431122536336e+01  -9.90565427543471e+01  -1.39863058867336e-01
  4.65786124809338e+01  -8.75491739319100e+01  -8.31330568526640e-01
  6.98483557326398e+01   7.01697991396945e+01  -9.91952297510380e-01
  1.00300699712769e+02   1.00332159684970e+01   8.01268769503155e-01
 -3.81620398811578e+01  -9.37184610893863e+01   1.19037126552055e+00
  6.71808214170653e+01  -7.31380598488546e+01  -6.90188980512821e-01
 -6.93656666953292e+01  -7.34435692615056e+01   1.02253996983577e+00
 -8.68907802573278e+00  -1.00138363868779e+02   5.14635726610647e-01
 -1.54970740996509e+01  -9.95112240797290e+01   7.10689718103463e-01
  1.00871352635049e+02  -3.43594046338378e-01   8.71937818666841e-01
  7.61410283583710e+01  -6.41485103161131e+01  -4.38523638674741e-01
  1.04765285994161e+01   9.88502232451412e+01  -5.96155571785715e-01
 -3.48813969981801e+01   9.44189108101853e+01   6.56060796786503e-01
 -5.97625774559531e+01   8.11622921878426e+01   7.91286018998287e-01
 -9.88117122094460e+01  -8.26016931845083e+00  -8.43634259156860e-01
 -8.43694402462738e+01  -5.43742481792253e+01   3.73110505381653e-01
  5.62436063288928e+01  -8.16718994397296e+01  -8.35276378276654e-01
 -8.51566669033422e+01   5.24931173171742e+01   3.59199675865786e-02
  5.39221823629581e+01  -8.32113580966188e+01  -8.44910029450134e-01
```