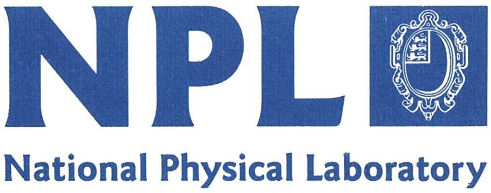
NPL Report DITC 228/94



Algorithms for geometric tolerance assessment

B P Butler, A B Forbes, and P M Harris

November 1994

NPL Report DITC 228/94

November 1994

Algorithms for geometric tolerance assessment

B P Butler, A B Forbes, and P M Harris
Division of Information Technology and Computing
National Physical Laboratory
Teddington
Middlesex
United Kingdom
TW11 0LW

ABSTRACT

Design and tolerance information can be stated in terms of nominal form, parameter constraints, and form constraints involving measured data points. The tolerance assessment problem is to find a set of parameter values that satisfy the constraints, and this may be achieved by formulating and solving an associated optimisation problem. In this report we describe how to exploit standard optimisation software in order to solve this optimisation problem. Our intention is to describe generic harnesses or templates that act as interfaces between the tolerance assessment problem and the optimisation software. The harnesses are specified using Matlab although, since their design is transparent, translation into other languages should be straightforward. We illustrate the application of these submodules to a number of typical tolerance assessment problems.

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

ILLUSTRATIONS

TABLES

1 INTRODUCTION

This report describes algorithms and software for geometric tolerance assessment. In an earlier report [18] we showed how a class of design and tolerance information can be represented in terms of parametrized curves and surfaces $\mathbf{a} \mapsto \mathcal{S}(\mathbf{a})$, parameter constraints

$$C(\mathbf{a}) \geq 0, \tag{1}$$

and form constraints

$$F(X;\mathbf{a}) \geq 0, \tag{2}$$

involving data points $X = \{\mathbf{x}_i : i \in I\}$ gathered by a coordinate measuring system (CMS) from the surface of the workpiece under assessment. The tolerance assessment problem is then: find parameters $\mathbf{a}$ (if they exist) to satisfy (1) and (2).

We consider the related optimisation problem

$$\max_{s,\mathbf{a}} s \quad \text{subject to} \quad \begin{array}{rcl} C(\mathbf{a}) & \geq & s, \\ F(X;\mathbf{a}) & \geq & s. \end{array} \tag{3}$$

The sign of s at the solution indicates if the part is within or out of tolerance: if $s \geq 0$, the part is within tolerance; if $s < 0$, the part is out of tolerance. The magnitude of s at the solution gives some measure of how far a part is within or out of tolerance, information which can be fed back into the control of the manufacturing process.

In this report we describe how to exploit standard optimisation software in order to solve the optimisation problem (3) in general and for a number of important particular cases outlined in [18]. Our aim is to describe generic harnesses or templates that act as interfaces between the tolerance assessment problem and the optimisation software. We indicate the computations performed by the harness modules and specify, in Matlab, templates for the modules. Anyone requiring further information about the software is invited to contact the authors.

In section 2 we summarise the types of algorithms which can be applied to tolerance assessment and in section 3 we give a general overview of how standard optimisation software can be used to solve the tolerance assessment problem. The form constraints (2) that appear in the optimisation problem (3) typically involve the distances from the measured data to the workpiece. Section 4 covers the necessary theory concerning distance functions and discusses algorithms for calculating distances from a point to a general curve or surface.

Sections 5–7 deal with various types of tolerance assessment problems which are common in practice. Our summary and concluding remarks are given in section 8.

2 TOLERANCE ASSESSMENT AND MATHEMATICAL PROGRAMMING

In this section we describe in very general terms algorithms which can be used to solve the tolerance assessment problem. It is not our goal to describe in detail the operation of these algorithms but rather we endeavour to give a flavour of how they operate so that users can be more confident in applying software implementations of these algorithms to geometric tolerancing problems.

2.1 CONSTRAINED OPTIMISATION

The tolerance assessment problem (3) is a type of constrained optimisation problem which can be expressed more generally as the minimisation of an objective function E with respect to a vector of parameters $\mathbf{b}$, subject to inequality constraints, i.e.

$$\min_{\mathbf{b}} E(\mathbf{b}) \quad \text{subject to} \quad c_i(\mathbf{b}) \geq 0, \quad i \in I. \tag{4}$$

From the theory for constrained optimisation it can be shown that, under some mild regularity and smoothness assumptions, the following *Kuhn-Tucker* conditions must necessarily hold at a solution $\mathbf{b}^*$:

There exist λ_i, $i \in I$, such that

$$\begin{aligned} \nabla E(\mathbf{b}^*) &= \sum_i \lambda_i \nabla c_i(\mathbf{b}^*), \\ c_i(\mathbf{b}^*) &\geq 0, \quad i \in I, \\ \lambda_i &\geq 0, \quad i \in I, \\ \lambda_i c_i(\mathbf{b}^*) &= 0, \quad i \in I. \end{aligned}$$

It is noted that these conditions involve the first derivatives of E and c_i with respect to the parameters b_j; there are also necessary conditions involving second derivatives.

Moreover, all that is claimed is that $\mathbf{b}^*$ is a solution to the general constrained optimisation problem that is *locally* best, i.e., there exists a neighbourhood of $\mathbf{b}^*$ in which for all points $\mathbf{b}$ that satisfy the inequality constraints, $E(\mathbf{b}) \geq E(\mathbf{b}^*)$. However, the solution $\mathbf{b}^*$ need not be unique: there may be several such local solutions, and the best of these is called the *global* solution. We discuss later the implications of non-uniqueness for the tolerance assessment problem.

Algorithms to solve the general constrained optimisation problem return a local solution. Typically, they start from an estimate of the solution, examine in what way the optimality conditions fail to hold, and then produce a better estimate, following the steps below:

Constrained optimisation algorithm (COA)

I Given an estimate $\mathbf{b}$ of the solution, determine a set of constraints $I^a \subset I$ which are likely to be *active* at the solution, i.e. those for which $c_i(\mathbf{b}^*) = 0$.

II Determine $\mathbf{p}$ which approximately solves

$$\min_{\mathbf{p}} E(\mathbf{b} + \mathbf{p})$$

subject to

$$c_i(\mathbf{b} + \mathbf{p}) = 0, \quad i \in I^a.$$

III Adjust $\mathbf{p}$ if necessary so that $\mathbf{b} + \mathbf{p}$ is judged to be a better estimate of the solution, and replace $\mathbf{b}$ by $\mathbf{b} + \mathbf{p}$.

These steps are repeated until suitable termination criteria are satisfied.

Step II is generally achieved by linearising the constraints using first derivative information and approximating the objective function $E(\mathbf{b})$ using first and (estimates of) second derivative information. Thus, this step will require the calculation of $E(\mathbf{b})$, $\nabla E(\mathbf{b})$, $c_i(\mathbf{b})$ and $\nabla c_i(\mathbf{b})$, $i \in I$, given an estimate $\mathbf{b}$ of the solution.

At step III it is important to find an acceptable balance between reducing the objective function E and reducing any constraint violations, i.e. those for which $c_i(\mathbf{b}) < 0$. This is usually done using a *merit function* $M(\mathbf{b})$ which depends on the objective function and constraints.

The improvement generated at each iteration depends crucially on the determination of the *active set* I^a at step I. If the correct set of active constraints is identified, the algorithm will usually converge quite quickly. However, for many problems there will be numerous changes in the active set and it is important that these changes are made in a systematic and constructive way.

If the constraints c_i are linear in the parameters $\mathbf{b}$ then steps I to III are much easier than for the nonlinear case. If, in addition, E is also linear then further simplifications are possible.

2.2 MINIMAX PROBLEMS

A special case of the general constrained optimisation problem is the minimax problem:

$$\min_{\mathbf{a}} \max_{i \in I} e_i(\mathbf{a}),$$

which can be reformulated as

$$\min_{s,\mathbf{a}} s \qquad \text{subject to} \qquad s - e_i(\mathbf{a}) \geq 0, \quad i \in I. \tag{5}$$

Thus, the minimax problem is to find the optimal $\mathbf{a}^*$, i.e., that which minimises the maximum of the functions $e_i(\mathbf{a})$. Clearly the COA can be applied to this problem. However, the particular structure of (5) can be exploited to produce a feasible descent algorithm (FDA) to solve the minimax problem:

Feasible descent algorithm (FDA)

I Given $\mathbf{a}$, set $s = \max_i e_i(\mathbf{a})$, and determine a set of constraints $I^a \subset I$ which are likely to be active at the solution with $\{i : e_i(\mathbf{a}) = s\} \subset I^a$.

II Determine p which approximately solves

$$\min_{s,\mathbf{p}} s$$

subject to

$$e_i(\mathbf{a} + \mathbf{p}) = s, \quad i \in I^a.$$

III Find $\alpha > 0$ such that

$$\max_i e_i(\mathbf{a} + \alpha\mathbf{p}) < \max_i e_i(\mathbf{a}).$$

These steps are repeated until suitable termination criteria are satisfied.

The main simplifications of this algorithm over the constrained optimisation algorithm given above are (i) the FDA algorithm requires only information about the constraints, namely e_i and ∇e_i, and (ii) in step III it is much easier to determine whether a step $\mathbf{p}$ leads to an improved estimate: there is no need for a merit function.

2.3 CHEBYSHEV APPROXIMATION PROBLEMS

A further specialisation of the minimax problem is the Chebyshev approximation problem:

$$\min_{\mathbf{a}} \max_{i \in I} |e_i(\mathbf{a})|,$$

which can be reformulated as

$$\min_{s,\mathbf{a}} s \quad \text{subject to} \quad \begin{array}{rcl} s + e_i(\mathbf{a}) & \geq & 0, \\ s - e_i(\mathbf{a}) & \geq & 0. \end{array} \tag{6}$$

A feasible descent algorithm can be applied to solve the Chebyshev approximation problem, usually with modifications to exploit the fact that the terms $e_i(\mathbf{a})$ appear twice. Of course, a general purpose constrained optimisation algorithm can also be employed.

2.4 TOLERANCE ASSESSMENT

Returning to the general tolerance assessment problem (3), we now provide a reformulation which aligns it more closely with the general constrained optimisation problem (4). We suppose that the parameter constraints (1) can be written as

$$\pi_i(\mathbf{a}) \geq 0, \quad i \in \Pi,$$

and that the form constraints (2) can be written as

$$\phi_i(X, \mathbf{a}) \geq 0, \quad i \in \Phi.$$

As the notation suggests, only the form constraints depend on the measured data X. Generally, $\phi_i(X, \mathbf{a})$ involves $d(\mathbf{x}, \mathbf{a})$, the distance from a point $\mathbf{x}$ to the surface $\mathcal{S}(\mathbf{a})$.

The tolerance assessment problem (3) can now be written as

$$\min_{s,\mathbf{a}} s \quad \text{subject to} \quad \begin{array}{rcll} s + \pi_i & \geq & 0, & i \in \Pi, \\ s + \phi_i & \geq & 0, & i \in \Phi. \end{array} \tag{7}$$

In this formulation, the part is within tolerance if $s \leq 0$, and the part is out of tolerance if $s > 0$. Furthermore, it is seen that tolerance assessment is a type of minimax problem (5). We shall also see that a number of particular problems can be posed as Chebyshev approximation problems (6).

As noted earlier, the constrained optimisation problem (7) may have several local solutions, and the particular solution returned by an algorithm (such as the COA or FDA) will depend on the starting estimate employed. For the tolerance assessment problems discussed in this work, because we often have knowledge about the size and position, etc., of the manufactured part(s) being considered, sensible initial estimates can be determined. Consequently, the global solution (the required solution) can normally be expected. However, if $s \leq 0$ at the solution found, it follows that the part is within tolerance even if the solution is locally best. Only if $s > 0$ at the solution can there be any doubt about the conclusion that the workpiece fails to meet its specification.

2.5 LEAST SQUARES TOLERANCE ASSESSMENT

Although there are algorithms with corresponding software implementations available to perform constrained optimisation it has to be realised that these optimisation problems are by no means easy. As indicated earlier, these algorithms are far from elementary and their successful implementation requires a detailed knowledge of a number of mathematical and computational disciplines. In addition, it would be unrealistic to expect them to perform satisfactorily on all problems.

For this reason, there can be considerable advantage in formulating tolerance assessment problems in ways which lead to easier optimisation problems. For example it is possible to replace a Chebyshev approximation problem (6) by an approximating least squares (Gaussian) problem:

$$\min_{\mathbf{a}} \sum_{i \in I} e_i^2(\mathbf{a}), \tag{8}$$

which can be solved using much simpler least squares optimisation algorithms. In the examples in later sections we will indicate where alternative least squares formulations can be introduced. Since a least squares solution will in general be different from a Chebyshev solution, the effect of this on the behaviour of the tolerance assessment process for a given application will need to be examined.

It is worth noting that if the solution produced by an approximate method, such as least squares tolerance assessment, satisfies all the form and parameter constraints then the part is within tolerance. Consequently, an approximate method will correctly identify a *subset* of the parts within tolerance.

2.6 NOTES AND REFERENCES

A general discussion of the geometric tolerance assessment problem is given in the earlier report [18]. For (much) more on optimisation, see for example [12, 13, 14, 20, 21, 26]. Algorithms for the minimax and Chebyshev problems are discussed in [5, 10, 25, 28, 31] (and elsewhere), and publicly available software for constrained optimisation includes [5, 22, 29] and [27, Chapter E04].

The numerical results presented in this report were generated using software written in Matlab [23] running on a dedicated PC with the following specification: 25MHz 386 processor with 387 maths co-processor. The particular optimisation software used was the Matlab function **minimax.m** included in the Matlab Optimisation Toolbox [22]. This routine implements a sequential quadratic programming method [6] with modifications to the line search and Hessian. At each major iteration of the algorithm, a quadratic program defined by the current estimate of the active constraints is solved (*cf* step II of the COA). For each example, we indicate the number of major iterations taken to generate a solution: in all cases a solution was generated in real-time in less than a minute. We note that the time to solve the quadratic program is dependent on the number of parameters for the problem and the size of the active set of constraints. The number of measured points influences the time to update an estimate of the active set (step I). However, this part of the computation is usually quite fast and, for this reason, there is little penalty, in terms of computation time, associated with processing many points, provided they can be measured. Moreover, the greater the number of measured points, the more confident we can be in the assessment of the workpiece.

3 TOLERANCE ASSESSMENT USING OPTIMISATION SOFTWARE

In this section we discuss in general terms how optimisation software can be used to solve the tolerance assessment problem. We suppose that we have available software to solve the general constrained optimisation problem (4) which requires the user to supply submodules to calculate the objective function E and constraint values c_i and their first derivatives ∇E and ∇c_i at a general point $\mathbf{b}$. Typically, the specifications for these modules would be similar to the following Matlab [23, 24] templates:

objfun.m

```
  function [ E, gradE ] = objfun(bb)
%
% Objective function and gradient evaluation.
%
% Input     bb  - n x 1 array of optimisation parameters.
%
% Output     E  -       value of the objective function E(bb)
%                       at bb.
%        gradE  - n x 1 array containing the gradient of E(bb).
```

confun.m

```
  function [ c, gradc ]  = confun(bb)
%
% Constraint function and gradient evaluation.
%
% Input     bb  - n x 1 array of optimisation parameters.
%
% Output     c  - m x 1 array containing the values of the constraint
%                       functions at bb.
%        gradc  - m x n array containing the gradients of c:
%                       gradc(i,j) = partial c(i)/ partial bb(j).
```

Our aim is to design *harnesses* in the form of generic submodules which can act as interfaces between the tolerancing assessment problems and the optimisation software. In subsequent sections we will give particularisations of these harnesses appropriate for the application considered. Often we will supply Matlab templates for these harness modules although, since their design is transparent, translation into other languages should be straightforward. In practice, the harness modules assemble information about the tolerance problem to produce objective function and constraint function evaluation routines.

3.1 OBJECTIVE FUNCTION MODULE

Comparing (7) with (4) we see that the optimisation parameters are given by $\mathbf{b}^{\mathrm{T}} = (s,\ \mathbf{a}^{\mathrm{T}})$ and the corresponding objective function is

$$E(s, \mathbf{a}) = s.$$

We note that the objective function is very simple and will be the same across different tolerance assessment applications, so that we can design a generic submodule **gtaobj.m** for evaluating the objective function and its gradient:

gtaobj.m

```
  function [ E, gradE ] = gtaobj(bb)
%
% Objective function and gradient evaluation for geometric
% tolerance assessment.
%
% Input     bb  - n x 1 array of optimisation parameters (s, aa)'.
%
% Output     E  -       value of the objective function = s.
%        gradE  - n x 1 array containing the gradient of the
%                       objective function.
%
%  [ E, gradE ] = gtaobj(bb)

% ------------------------------------------------------------------------
%
       n = length(bb);
%
       E = bb(1);
   gradE = [ 1; zeros(n-1, 1) ];
%
% ------------------------------------------------------------------------
```

3.2 CONSTRAINT EVALUATION MODULE

The constraints $c_i(\mathbf{b})$, $i \in I = \Pi \cup \Phi$, are partitioned into the parameter and form constraints expressed as in (7), so that

$$c_i(\mathbf{b}) = \begin{cases} s + \pi_i(\mathbf{a}), & i \in \Pi, \\ s + \phi_i(X;\mathbf{a}), & i \in \Phi. \end{cases} \tag{9}$$

From this we see that

$$\frac{\partial c_i}{\partial s} = 1, \quad i \in I,$$

and

$$\nabla_{\mathbf{a}} c_i = \begin{cases} \nabla_{\mathbf{a}} \pi_i, & i \in \Pi, \\ \nabla_{\mathbf{a}} \phi_i, & i \in \Phi. \end{cases}$$

Thus, we can write a generic module for evaluating the constraint functions and their gradients based on the following template:

gtacon.m

```
  function [ c, gradc ]  = gtacon(X, bb)
%
% Constraint function and gradient evaluation for geometric
% tolerance assessment.
%
% Input      X  - mX x q array of data points (q = 2 or 3).
%           bb  -  n x 1 array of optimisation parameters (s, aa)'.
%
% Output     c  - mc x 1 array containing the value of the constraint
%                        functions at bb.
```

```
%        gradc  - mc x n array containing the gradients of c:
%                          gradc(i,j) = partial c(i)/ partial bb(j).
%
%  [ c, gradc ]  = gtacon(X, bb)

% ------------------------------------------------------------------------
%
     n = length(bb);
     s = bb(  1);
    aa = bb(2:n);
%
% Calculate parameter constraint values and gradients.
%
     [ pi, gradpi  ] = parcon(   aa);
%
% Calculate form      constraint values and gradients.
%
     [phi, gradphi ] = forcon(X, aa);
%
% Assemble constraint values.
%
        c = [s+pi; s+phi];
%
% Assemble constraint gradients.
%
       mc = length(c);
%
    gradc = [ ones(mc, 1)  [ gradpi; gradphi ] ];
%
% ------------------------------------------------------------------------
```

Here, the submodules **parcon.m** and **forcon.m** calculate π_i and $\nabla_{\mathbf{a}}\pi_i$, $i \in \Pi$, and ϕ_i and $\nabla_{\mathbf{a}}\phi_i$, $i \in \Phi$, respectively.

3.3 MULTI-COMPONENT PROBLEMS

For many tolerancing problems, the curve or surface $\mathbf{a} \mapsto \mathcal{S}(\mathbf{a})$ will contain a number n_K of features or subcomponents $\mathbf{a}_k \mapsto \mathcal{S}_k(\mathbf{a}_k)$. Correspondingly, the measured data points X will be partitioned into subsets $X_k = \{\mathbf{x}_i : i \in I_k\}$, $k = 1, \ldots, n_K$, with X_k representing points on the kth feature. As a result the form constraints ϕ_i will also need to be partitioned and the constraint function and gradient evaluation may require different submodules according to which feature is being evaluated. In practice, therefore, the module **forcon.m** will execute a sequence of steps:

for k $= 1$ **to** n_k

I$_k$ Extract X_k, $\mathbf{a}_k$ from X and $\mathbf{a}$.

II$_k$ Call submodule **forcon**$_k$**.m** to calculate ϕ_i and $\nabla_{\mathbf{a}_k}\phi_i$, $i \in I_k$.

next k;

With multi-component tolerance assessment the matrix of constraint gradients $J_{\mathbf{c}} = \nabla_{\mathbf{a}}\mathbf{c}$ will generally have a block structure [11] dictated by which set of constraints depend on which

subset of the parameters. For large problems it may be beneficial or essential to exploit this structure. We will be content for the most part to treat all matrices as full.

4 DISTANCE FUNCTIONS

For form constraints it is necessary to calculate the distance $d = d(\mathbf{x}, \mathbf{a})$ from a point $\mathbf{x}$ to a parametrized curve or surface $\mathcal{S}(\mathbf{a})$ and its derivatives with respect to $\mathbf{a}$. For simple curves and surfaces, it is often possible to write down an explicit formula for the distance function. For example, if a circle is parametrized by its centre coordinates (a, b) and its radius r_0, the distance from a point (x, y) to the circle is given by the formula

$$d = d(x, y, a, b, r_0) = [(x-a)^2 + (y-b)^2]^{1/2} - r_0. \tag{10}$$

For general curves and surfaces no analytical formula can be derived and an algorithmic approach has to be adopted. Notice that we are concerned here with the *signed* distance of a point from a curve or surface: thus, the use of (10) means that a point inside the circle is assigned a negative distance, and a point outside the circle a positive distance.

Suppose $\mathbf{x}$ is a point near $\mathcal{S}(\mathbf{a})$ and we want to calculate the distance $d = d(\mathbf{x}, \mathbf{a})$ from $\mathbf{x}$ to $\mathcal{S}(\mathbf{a})$. If $\mathbf{x}$ is sufficiently close to $\mathcal{S}$, this distance will be realised as

$$d(\mathbf{x}, \mathbf{a}) = \|\mathbf{x} - \mathbf{x}^*(\mathbf{x}, \mathbf{a})\| = \langle \mathbf{x} - \mathbf{x}^*(\mathbf{x}, \mathbf{a}),\ \mathbf{x} - \mathbf{x}^*(\mathbf{x}, \mathbf{a}) \rangle^{1/2}, \tag{11}$$

where $\mathbf{x}^*(\mathbf{x}, \mathbf{a})$ is the point on $\mathcal{S}(\mathbf{a})$ closest to $\mathbf{x}$. As indicated by the notation, $\mathbf{x}^*(\mathbf{x}, \mathbf{a})$ is a function of the surface parameters $\mathbf{a}$ and the point $\mathbf{x}$. Differentiation gives

$$\frac{\partial d}{\partial a_j} = -\langle \frac{\partial \mathbf{x}^*}{\partial a_j}(\mathbf{x}, \mathbf{a}),\ \mathbf{n}^* \rangle, \tag{12}$$

where $\mathbf{n}^*$ is the unit normal to the surface $\mathcal{S}$ at $\mathbf{x}^*$. Thus, to invoke the optimisation algorithms we need to be able to calculate $\mathbf{x}^*$, $\mathbf{n}^*$ and the derivatives of $\mathbf{x}^*$ with respect to the surface parameters.

Two cases need to be considered:

(i) The surface $\mathcal{S}$ is given explicitly in the form

$$\mathbf{x} \in \mathcal{S} \iff \mathbf{x} = \mathbf{f}(\mathbf{u}, \mathbf{a}). \tag{13}$$

Here, $\mathbf{u}$ are *auxiliary* parameters that are mapped onto points $\mathbf{x}$ in $\mathcal{S}$ and are distinct from the *surface* parameters $\mathbf{a}$ defining the form of $\mathcal{S}$.

(ii) $\mathcal{S}$ is given implicitly in the form

$$\mathbf{x} \in \mathcal{S} \iff f(\mathbf{x}, \mathbf{a}) = 0. \tag{14}$$

Case (i) covers functional descriptions of the form $z = f(x, y, \mathbf{a})$ as well as general parametric curves and surfaces. This is because points (x, y, z) in a surface for which $z = f(x, y, \mathbf{a})$ may be specified by

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} u \\ v \\ f(u, v, \mathbf{a}) \end{bmatrix},$$

which is in the form of (13). The two types of representation are illustrated below using the example of a circle: see (19) and (20), respectively.

4.1 EXPLICIT SURFACES

4.1.1 Distance evaluation

The distance from a point $\mathbf{x}$ to a surface $\mathcal{S}(\mathbf{a})$ given explicitly by (13) can be determined by finding $\mathbf{u}^* = \mathbf{u}^*(\mathbf{x}, \mathbf{a})$ which solves the minimisation problem

$$\min_{\mathbf{u}} \langle \mathbf{x} - \mathbf{f}(\mathbf{u}, \mathbf{a}),\ \mathbf{x} - \mathbf{f}(\mathbf{u}, \mathbf{a}) \rangle. \tag{15}$$

This is an unconstrained optimisation problem and can be solved by a number of techniques including variants of Newton's method.

4.1.2 Gradient of the distance function

For a surface given by (13),

$$\mathbf{x}^*(\mathbf{x}, \mathbf{a}) = \mathbf{f}(\mathbf{u}^*(\mathbf{x}, \mathbf{a}), \mathbf{a}),$$

so that

$$\frac{\partial \mathbf{x}^*}{\partial a_j} = \sum_k \frac{\partial \mathbf{f}}{\partial u_k} \frac{\partial u_k}{\partial a_j} + \frac{\partial \mathbf{f}}{\partial a_j}.$$

At $\mathbf{u}^*$ the vectors $\partial \mathbf{f}/\partial u_k$ are tangential to the surface, and hence orthogonal to $\mathbf{n}^*$, so that from (12),

$$\frac{\partial d}{\partial a_j} = -\langle \frac{\partial \mathbf{f}}{\partial a_j}, \mathbf{n}^* \rangle. \tag{16}$$

Thus, in order to calculate the derivatives of the distance function it is necessary to calculate the optimal $\mathbf{u}^*$, the normal $\mathbf{n}^*$ and derivatives $\partial \mathbf{f}/\partial a_j$ evaluated at $\mathbf{u}^*$; although $\mathbf{u}^*$ depends on $\mathbf{a}$, it is not necessary to calculate the derivatives of $\mathbf{u}^*$.

4.2 IMPLICIT SURFACES

4.2.1 Distance evaluation

We now consider surfaces $\mathcal{S}(\mathbf{a})$ given implicitly by (14). The distance from a point $\mathbf{x}$ to such a surface can be determined by finding $\mathbf{u}^* = \mathbf{u}^*(\mathbf{x}, \mathbf{a})$ which solves

$$\min_{\mathbf{u}} \langle \mathbf{x} - \mathbf{u},\ \mathbf{x} - \mathbf{u} \rangle \quad \text{subject to} \quad f(\mathbf{u}, \mathbf{a}) = 0. \tag{17}$$

Newton's method can again be used to solve this problem.

4.2.2 Gradient of the distance function

For a surface given by (14), we have simply

$$\mathbf{x}^*(\mathbf{x}, \mathbf{a}) = \mathbf{u}^*(\mathbf{x}, \mathbf{a}).$$

Differentiating $f(\mathbf{x}^*(\mathbf{x}, \mathbf{a}), \mathbf{a}) = 0$ with respect to a_j,

$$\langle \nabla_{\mathbf{x}} f, \frac{\partial \mathbf{x}^*}{\partial a_j} \rangle + \frac{\partial f}{\partial a_j} = 0.$$

Noting that $\mathbf{n}^* = \nabla_{\mathbf{x}} f / \|\nabla_{\mathbf{x}} f\|$ and comparing the above equation with (12), we see that

$$\frac{\partial d}{\partial a_j} = \frac{\partial f}{\partial a_j} / \|\nabla_{\mathbf{x}} f\|. \tag{18}$$

As in the case of explicitly defined surfaces, to calculate the distance function and its gradient, it is necessary to calculate the optimal $\mathbf{x}^*$ and the derivatives of f evaluated at $(\mathbf{x}^*, \mathbf{a})$; although $\mathbf{x}^*$ is a function of $\mathbf{a}$, it is not necessary to calculate $\partial \mathbf{x}^* / \partial a_j$.

Example: circle gradient calculations. Differentiating d given in equation (10) gives

$$\begin{aligned} \frac{\partial d}{\partial a} &= -\frac{x-a}{r}, \\ \frac{\partial d}{\partial b} &= -\frac{y-b}{r}, \\ \frac{\partial d}{\partial r_0} &= -1, \end{aligned}$$

where

$$r = [(x-a)^2 + (y-b)^2]^{1/2}.$$

Now, we can also regard a circle as given parametrically by

$$u \mapsto (f(u,a,b,r_0),\ g(u,a,b,r_0) = (a + r_0 \cos u,\ b + r_0 \sin u). \tag{19}$$

We then use equation (16) to calculate the gradient in terms of the optimal u^* and normal vector $(\cos u^*, \sin u^*)$:

$$\begin{aligned} \frac{\partial d}{\partial a} &= -\frac{\partial f}{\partial a}\cos u^* - \frac{\partial g}{\partial a}\sin u^* \\ &= -\cos u^*, \\ \frac{\partial d}{\partial b} &= -\frac{\partial f}{\partial b}\cos u^* - \frac{\partial g}{\partial b}\sin u^* \\ &= -\sin u^*, \\ \frac{\partial d}{\partial r_0} &= -\frac{\partial f}{\partial r_0}\cos u^* - \frac{\partial g}{\partial r_0}\sin u^*, \\ &= -\cos^2 u^* - \sin^2 u^* = -1. \end{aligned}$$

Agreement with the previous calculations follows from the fact that for a circle given by (10), $u^*(x,y,a,b,r_0) = \arctan \frac{y-b}{x-a}$.

Finally we can describe a circle implicitly by

$$f(x,y,a,b,r_0) = x^2 + y^2 - 2ax - 2by + a^2 + b^2 - r_0^2 = 0, \tag{20}$$

and use equation (18) to calculate derivatives once (x^*, y^*) has been found:

$$\begin{aligned} \frac{\partial d}{\partial a} &= \frac{2a - 2x^*}{[(2a-2x^*)^2 + (2b-2y^*)^2]^{1/2}} \\ &= -\frac{x^* - a}{r_0} = -\cos u^*, \\ \frac{\partial d}{\partial b} &= -\sin u^*, \\ \frac{\partial d}{\partial r_0} &= \frac{-2r_0}{2r_0} = -1. \end{aligned}$$

4.3 CALCULATING GRADIENTS WITH RESPECT TO OTHER PARAMETERS

For some problems, the point $\mathbf{x}$ near the surface $\mathcal{S}$ depends on parameters $\mathbf{t}$ and it is also necessary to calculate $\nabla_{\mathbf{t}} d(\mathbf{x}(\mathbf{t}), \mathbf{a})$. In this situation the point $\mathbf{x}^*(\mathbf{x}(\mathbf{t}), \mathbf{a})$ on the surface $\mathcal{S}(\mathbf{a})$ nearest $\mathbf{x}(\mathbf{t})$ is a function of $\mathbf{t}$ (and $\mathbf{a}$), and

$$\begin{aligned} \frac{\partial d}{\partial t_j} &= \frac{\partial}{\partial t_j} \|\mathbf{x}(\mathbf{t}) - \mathbf{x}^*(\mathbf{x}(\mathbf{t}), \mathbf{a})\|, \\ &= \langle \frac{\partial \mathbf{x}}{\partial t_j} - \sum_k \frac{\partial \mathbf{x}^*}{\partial x_k} \frac{\partial x_k}{\partial t_j}, \mathbf{n}^* \rangle. \end{aligned}$$

The vectors $\partial \mathbf{x}^* / \partial x_k$ are tangent to the surface and therefore $\langle \frac{\partial \mathbf{x}^*}{\partial x_k}, \mathbf{n}^* \rangle = 0$. It follows that

$$\frac{\partial d}{\partial t_j} = \langle \frac{\partial \mathbf{x}}{\partial t_j}, \mathbf{n}^* \rangle. \tag{21}$$

Thus, these calculations involve only the determination of $\mathbf{x}^*$ and the corresponding normal $\mathbf{n}^*$ at $\mathbf{x}^*$ and the derivatives of $\mathbf{x}$ with respect to $\mathbf{t}$; although $\mathbf{x}^*$ depends on $\mathbf{t}$ and $\mathbf{a}$, it is not necessary to calculate the derivatives of $\mathbf{x}^*$ with respect to these parameters.

4.4 CALCULATING NEAREST POINTS AND NORMALS

We have seen above that the calculation of distance functions and gradients for general curves and surfaces relies on the determination of nearest points $\mathbf{x}^*(\mathbf{x}, \mathbf{a})$. For the cases considered, the optimal $\mathbf{x}^*$ can be found by solving a nonlinear minimisation problem ((15) or (17)) in one or two variables. We have performed many numerical experiments using different algorithms and these experiments confirm the point more generally true in function minimisation that Newton methods employing second derivative information (and safeguarded against divergence) are very successful, particularly when started at a point close to the required nearest point. It can be argued that they are particularly suited to distance minimisation. For applications in tolerance assessment we expect (i) sensible parametrizations of the surfaces, (ii) the points to be close to the surface relative to its curvature and (iii) to have good estimates of the parameters. We require accurate estimates of the optimal $\mathbf{x}^*$ and that these estimates should be calculated efficiently. Under these circumstances the distance function is very well behaved with a well defined minimum and the fast convergence of a Newton method is almost guaranteed.

4.5 DISTANCE FUNCTIONS AND TOLERANCE ASSESSMENT

The main point of the analysis above is to show that tolerance assessment for general curves and surfaces can proceed in much the same way as for simple curves and surfaces. To exploit optimisation software it is required to calculate constraint gradients. For tolerance assessment, the form constraints often involve distance functions and so it is necessary to calculate gradients of distance functions. From formulæ (16), (18) and (21), it is seen that once nearest points $\mathbf{x}^*$ have been found the calculation of *gradients* of distance functions for general curves and surfaces is no more complicated than for curves and surfaces for which explicit formulæ for gradients can be analytically derived.

We shall assume that distance function and gradient evaluations can be supplied in the form of a module specified as in the following template:

ge_dgd.m

```
  function [ d, gradd ]  = ge_dgd(X, aa, p1, p2, ... )
%
% Distance function and gradient evaluation.
%
% Input    X  -  m x q array of points, q = 2 or 3.
%         aa  -  n x 1 array of parameters defining the curve
%                   or surface.
%  p1,p2,...  -        additional parameters.
%
% Output
%          d  -  m x 1 array of distances d(x_i, aa).
%      gradd  -  m x n array of gradients.
```

For general curves and surfaces, it will also be necessary to store and update the auxiliary parameters $\mathbf{u}_i$ which determine the closest points $\mathbf{x}^*(\mathbf{x}_i, \mathbf{a})$. We will assume that this information is passed to the module by means other than through the parameter list using for example, common blocks, global variables or auxiliary files. With this strategy, the tolerance assessment software can be designed to cope equally with simple geometric elements and general curves and surfaces. The only differences between the two types of element occur at the low level modules **ge_dgd.m** which are already element specific.

With this approach we can be confident that the algorithms and software described below can be applied to a wide range of problems and we concentrate here mainly on aspects which relate to tolerance assessment *per se* rather than aspects which concern particular types of curve or surface.

4.6 SECOND DERIVATIVES OF DISTANCE FUNCTIONS

While gradients of distance functions do not require any derivative calculations involving $\mathbf{x}^*(\mathbf{x}, \mathbf{a})$, second derivatives do. Fortunately, most optimisation software requires first derivatives only and uses finite differences or other methods to estimate second order information. If necessary finite differencing software can be designed to take into account the auxiliary parameters $\mathbf{u}_i$ for calculations involving distances.

4.7 NOTES AND REFERENCES

Problems of form and tolerance assessment associated specifically with parametric curves and surfaces are considered in [8].

5 GEOMETRIC ELEMENTS

In this section, we look at geometric tolerance assessment problems associated with the simple geometric elements, primarily lines, planes, circles, spheres, cylinders, and cones and combinations of these elements. There are three types of tolerance assessment problems which are commonly associated with these elements, and we consider these in turn.

Zone tolerances. Zone tolerances specify the maximum deviation τ from nominal form:

Given a surface S of nominal form $\mathcal{S}(\mathbf{a})$ and a tolerance τ, for the measured surface to be within tolerance there must exist $\mathbf{a}$ such that for all $\mathbf{x} \in S$

$$|d(\mathbf{x};\mathbf{a})| \leq \tau.$$

Given a set of data points $X = \{\mathbf{x}_i : i \in I\}$ lying on the surface of S the related optimisation problem can be expressed as the Chebyshev approximation problem

$$\min_{\mathbf{a}} \max_{i \in I} |d(\mathbf{x}_i;\mathbf{a})|,$$

which can be reformulated as

$$\min_{s,\mathbf{a}} s \quad \text{subject to} \quad \left.\begin{array}{rcl} s + d(\mathbf{x}_i;\mathbf{a}) & \geq & 0, \\ s - d(\mathbf{x}_i;\mathbf{a}) & \geq & 0, \end{array}\right\} \quad i \in I. \tag{22}$$

If s at the solution is less than τ, the tolerance has been met.

The zone tolerance assessment problem has no parameter constraints and the form constraints are partioned into two sets

$$\left.\begin{array}{rcl} c^+(\mathbf{x}_i;\mathbf{b}) & = & s + d(\mathbf{x}_i;\mathbf{a}), \\ c^-(\mathbf{x}_i;\mathbf{b}) & = & s - d(\mathbf{x}_i;\mathbf{a}), \end{array}\right\} \quad i \in I, \tag{23}$$

with

$$\begin{array}{rcl} \dfrac{\partial c_i^{\pm}}{\partial s} & = & 1, \\ \nabla_{\mathbf{a}} c_i^{\pm} & = & \pm \nabla_{\mathbf{a}} d_i. \end{array}$$

The Matlab function **zotcon.m**, the specification of which is given below, evaluates the zone tolerance constraint functions (23) and their gradients given a module **ge_dgd.m** which calculates the distance functions d_i and gradients $\nabla_{\mathbf{a}} d_i$.

zotcon.m

```
  function [ c, gradc ] = zotcon(X,bb,ge_dgd,p1,p2,p3,p4,p5,p6)
%
% Constraint function and gradient evaluation for the
% zone tolerance assessment problem.
%
% Input
%           X  - m x q array of data points, q = 2 or 3.
%          bb  - n x 1 array of parameter values (s aa)'.
%      ge_dgd  - function to evaluate distances from points
%                to the geometric element and their gradients,
%                specified by
%
%                  [ d, gradd ] = ge_dgd(X, aa, p1, p2, ...)
%
%     p1,p2,.. - up to six additional parameters to be passed to
%                the function ge_dgd. <Optional>.
%
% Output    c  - (2*m) x 1 array of constraint values.
%        gradc - (2*m) x n array of constraint gradients.
%
% [ c, gradc ] = zotcon(X,bb,'ge_dgd',<p1,p2,p3,p4,p5,p6>)
```

Inscribed tolerances. Inscribed tolerances are used when it is required to specify that a manufactured part should contain an element of a prescribed size. The geometric elements for which these problems are usually defined include the circle in a fixed plane, sphere and cylinder.

For example, suppose we wish to know whether it is possible to fit within a part (that is nominally circular) a circle of prescribed radius R. The part is within tolerance if there exists a point $\mathbf{a}$ for which for all points $\mathbf{x}$ in the surface of the part, the distance from $\mathbf{x}$ to $\mathbf{a}$ is at least R.

The associated optimisation problem is:

Given data points $X = \{\mathbf{x}_i : i \in I\}$,

$$\max_{s,\mathbf{a}} s$$

subject to

$$r(\mathbf{x}_i, \mathbf{a}) \geq s, \quad i \in I,$$

where $r(\mathbf{x}, \mathbf{a})$ is, for example, the distance of the point $\mathbf{x}$ to the circle or sphere centre or cylinder axis. To apply the optimisation software more directly, we reformulate the *maximum inscribed* problem as

$$\min_{s,\mathbf{a}} s \quad \text{subject to} \quad s + r(\mathbf{x}_i, \mathbf{a}) \geq 0, \quad i \in I. \tag{24}$$

Circumscribed tolerances. Corresponding to inscribed tolerances, circumscribed tolerances are used when it is required to specify that a manufactured part should be contained within an element of a fixed size.

For example, suppose we wish to know whether it is possible to fit a part (that is nominally circular) within a circle of prescribed radius R. The part is within tolerance if there exists a point $\mathbf{a}$ for which for all points $\mathbf{x}$ in the surface of the part, the distance from $\mathbf{x}$ to $\mathbf{a}$ is no greater than R.

Circumscribed tolerances lead to *minimum circumscribed* problems of the form

$$\min_{s,\mathbf{a}} s \quad \text{subject to} \quad s - r(\mathbf{x}_i, \mathbf{a}) \geq 0, \quad i \in I. \tag{25}$$

It is quite straightforward to modify the template **zotcon.m** to obtain constraint function and gradient evaluation modules for the maximum inscribed and minimum circumscribed problems.

Example: minimum zone, maximum inscribed and minimum circumscribed circles. Consider the data shown in Figure 1 that represents a nominally circular artefact. The data was generated by randomly perturbing points sampled uniformly around the unit circle. We wish to know (a) whether the artefact lies within a zone tolerance of $\tau = 0.05$ from a circle, (b) whether a circular plug of radius 0.9 would fit inside the artefact, and (c) whether the artefact would fit inside a circular hole of radius 1.1.

To answer these questions we solve instances of the optimisation problems given above:

Minimum zone:

$$\min_{s,a,b,r_0} s \quad \text{subject to} \quad \left.\begin{array}{rcl} s + (r_i - r_0) & \geq & 0, \\ s - (r_i - r_0) & \geq & 0, \end{array}\right\} \quad i \in I.$$

Maximum inscribed:

$$\min_{s,a,b} s \quad \text{subject to} \quad s + r_i \geq 0, \quad i \in I.$$

Minimum circumscribed:

$$\min_{s,a,b} s \quad \text{subject to} \quad s - r_i \geq 0, \quad i \in I.$$

In each case r_i is the distance from the point $\mathbf{x}_i$ to the circle centre (a, b). We then compare the value of the objective function s at the solution with the specified tolerance. In order to solve these problems using general purpose optimisation software, we must supply a module for evaluating the distance functions and their derivatives with respect to the circle parameters.

For the minimum zone problem, $d_i = r_i - r_0$ and s measures the maximum departure in absolute value of the data from a circle with parameters (a, b, r_0). A Matlab module **ci_dgd.m**, the header for which is listed below, provides the required distance and gradient information by implementing the formulæ given in section 4.

ci_dgd.m

```
   function [d, gradd] = ci_dgd(X, aa)
%
%  Distance function and gradient evaluation for a circle in the plane
%  parametrized by circle centre coordinates (a, b) and radius r0.
%
%  Input    X  - m x 2 array of x and y coordinates.
%           aa  - 3 x 1 array of parameters (a, b, r0)'.
%
%  Output   d  - m x 1 array of distances.
%        gradd  - m x 3 array of gradients.
```

This module is used in conjunction with **zotcon.m** to evaluate the constraint functions and their gradients for the zone tolerancing problem:

```
[c, gradc] = zotcon(X, [s; aa], 'ci_dgd');
```

For the maximum inscribed and minimum circumscribed problems, $d_i = r_i$, and $|s|$ is, respectively, the radius of the maximum inscribed and minimum circumscribed circle with centre (a, b). A simple modification to **ci_dgd.m** allows us for these problems to calculate r_i and its derivatives with respect to a and b.

It is also necessary to provide initial estimates for the optimisation parameters. This can be done, for example, by solving a simpler approximation problem. For instance, a linear least squares circle fitting algorithm can be used to give estimates for (a, b) and r_0 (see Section 5.3 and [4, 16]). Here, we have chosen to use the centroid of the data to estimate (a, b) and, for the minimum zone problem, the mean of the distances of the closest and furthest points to this centre as an estimate of r_0.

In Table 1 we list, for each problem, the values of the parameters and the objective function s for our initial estimate and for the solution. The solution circles are illustrated in Figure 1. We conclude from these results that the zone tolerance has *not* been met by the artefact. However, the radius of the maximum inscribed circle exceeds 0.9 and so the given plug will fit inside the measured artefact. Similarly, the artefact will fit inside a circular hole of radius 1.1.

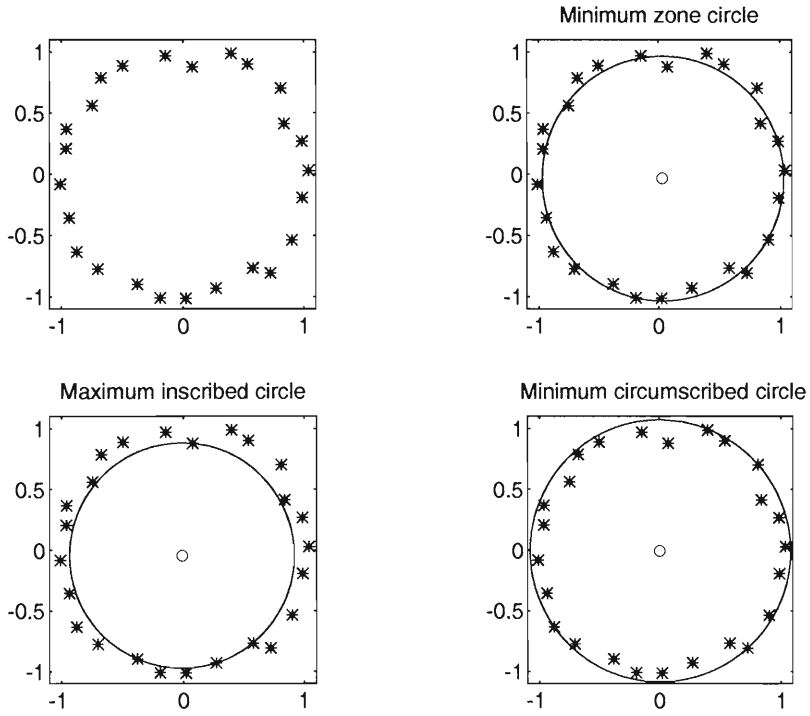


Figure 1 Minimum zone, maximum inscribed and minimum circumscribed circles for given data.

Table 1 Details of initial estimates and solutions for the minimum zone, maximum inscribed and minimum circumscribed circle problems. Number of major iterations for each problem is four, four and three, respectively.

Problem	Parameters	Initial estimate	Solution
Minimum	a	0.004224	0.029892
Zone	b	-0.001528	-0.034581
	r_0	0.983930	1.000765
	s	0.100328	0.085868
Maximum	a	0.004224	-0.008795
Inscribed	b	-0.001528	-0.045179
	s	-0.883602	-0.928214
Minimum	a	0.004224	0.001737
Circumscribed	b	-0.001528	-0.007156
	s	1.084258	1.078960

Example: circle with zone and radial tolerances. The previous example illustrated how we may approach tolerance assessment problems involving only form constraints. We now consider a problem that includes constraints on the parameters as well. Given measured data $\{\mathbf{x}_i : i \in I\}$, we wish to determine whether there exists a circle $C(a, b, r_0)$ satisfying the form constraints

$$|d_i| \leq \tau_f, \quad i \in I,$$

and the parameter constraint

$$|r_0 - R| \leq \tau_r.$$

Here, d_i is the distance from the point $\mathbf{x}_i$ to the circle C, and τ_f is a tolerance that specifies the maximum allowable departure of the artefact from a circle. The parameter constraint requires that the radius of the circle is within a tolerance τ_r of a given nominal value R.

The corresponding optimisation problem (recall (7)) that we solve is:

$$\min_{s,a,b,r_0} s$$

subject to

$$s + \tau_f - |d_i| \geq 0, \quad i \in I, \tag{26}$$
$$s + \tau_r - |r_0 - R| \geq 0. \tag{27}$$

Then, if $s \leq 0$ at the solution, a circle C can be found to satisfy the form and parameter constraints.

To solve this optimisation problem, we must provide a module for evaluating the constraint functions (26) and (27), and their derivatives with respect to s, a, b and r_0. The generic module **gtacon.m** described in section 3.2 provides us with a template for performing these calculations. It requires that we supply submodules in place of **forcon.m** and **parcon.m** to evaluate, respectively, the functions

$$\phi_i^{\pm} = \tau_f \pm d_i, \quad i \in I,$$
$$\pi^{\pm} = \tau_r \pm (r_0 - R),$$

and their derivatives. The following Matlab functions, **ci_for.m** and **ci_par.m**, perform these calculations using the function **ci_dgd.m** described before.

ci_for.m

```
   function [phi, gradphi] = ci_for(X, aa, tauf)
%
%  Geometric tolerancing for the circle with zone and
%  radial tolerances: evaluation of functions relating
%  to the form constraints and their gradients with
%  respect to the circle parameters.
%
%  Input      X  - m x 2 array of x and y coordinates.
%            aa  - 3 x 1 array of parameters (a, b, r0)'.
%          tauf  - Zone tolerance.
%
%  Output   phi  - (2*m) x 1 array of function values relating
%                    to the form constraints.
%       gradphi  - (2*m) x 3 array of gradients.
```

```
%
%  [phi, gradphi] = ci_for(X, aa, tauf)

%  ------------------------------------------------------------------------
%
%  Evaluate distances of the data to the given circle and
%  the derivatives of these distances with respect to the
%  circle parameters ...
%
   [d, gradd] = ci_dgd(X, aa);
%
%  Construct function values ...
%
   phi = [tauf-d; tauf+d];
%
%  ... and their gradients ...
%
   gradphi = [ -gradd; gradd ];
%
%  ------------------------------------------------------------------------
```

ci_par.m

```
   function [pi, gradpi] = ci_par(aa, Rad, tauR)
%
%  Geometric tolerancing for the circle with zone and
%  radial tolerances: evaluation of functions relating
%  to the radius constraint and their gradients with
%  respect to the circle parameters.
%
%  Input     aa  - 3 x 1 array of parameters (a, b, r0)'.
%            Rad  - Nominal value of circle radius.
%           tauR  - Radial tolerance.
%
%  Output    pi  - 2 x 1 array of function values relating
%                    to the radius constraint.
%         gradpi  - 2 x 3 array of gradients.
%
%  [pi, gradpi] = ci_par(aa, Rad, tauR)

%  ------------------------------------------------------------------------
%
%  Construct function values ...
%
   pi = [ (tauR - (aa(3) - Rad))
          (tauR + (aa(3) - Rad)) ];
%
%  ... and their gradients ...
%
   gradpi = [ 0 0 -1
              0 0  1 ];
%
%  ------------------------------------------------------------------------
```

To illustrate the use of these routines, consider again the data shown in Figure 1, and suppose we set $\tau_f = 0.1, R = 0.975$ and $\tau_r = 0.005$. We note from the results given in Table 1 that the

Table 2 Details of the initial estimate and solution to the geometric tolerance assessment problem for the circle with zone and radial tolerances. Number of major iterations to find the solution is three.

	Initial estimate	Solution
a	0.004224	0.001737
b	-0.001528	-0.007156
r_0	0.975000	0.979480
s	0.009258	-0.000520
$\max_i \lvert d_i \rvert$	0.109258	0.099480

minimum zone circle for the data, although satisfying the form constraints, does not satisfy the parameter constraint defined by R and τ_r.

In Table 2 we list the parameter values for an initial estimate of C, and for the solution returned by the optimisation software. For the initial estimate, we have chosen the centroid of the data to define the centre parameters (a, b), and we have set $r_0 = R$. For this circle, the form constraints are violated, and consequently $s > 0$. However, at the solution $s \leq 0$, and we conclude that a circle satisfying the specified zone and radial tolerances does exist.

Example: multi-component "capsule". In this example we illustrate how we may solve geometric tolerance assessment problems for multi-component artefacts composed of simple elements. We assume that individual modules **ge_dgd.m** specific to each simple element are available for the evaluation of distances and their gradients (see section 4.5).

The artefact we consider is a "capsule" formed from a cylinder of finite length which has attached at each end a hemi-spherical cap of the same radius as the cylinder. Given data points $\{\mathbf{x}_i : i \in I\}$ gathered from the surface of a capsule, and partitioned according to which element each point relates, we wish to solve the zone tolerancing assessment problem for this artefact.

Our first requirement is to parametrise the artefact. There are many possible ways of doing this, some of which will be numerically "better" than others. We have chosen to use parameters $(\mathbf{x}_0, \mathbf{a}, h_1, h_2, r_0)$ that describe the capsule in the following way:

- the cylindrical part of the capsule is defined by $\mathbf{x}_0$, a point on its axis, $\mathbf{a}$, a vector parallel to its axis, and r_0, its radius;
- each hemi-spherical cap $\mathcal{S}_i$, $i = 1, 2$, is defined by its centre $\mathbf{x}_0 + h_i\mathbf{a}$ and radius r_0.

In order that a capsule is represented uniquely using the chosen parametrisation, we require that $\mathbf{x}_0$ is a point in the xy-plane, $\mathbf{x}_0 := (x_0, y_0, 0)$, and we normalise the direction vector $\mathbf{a}$ so that $\mathbf{a} := (a, b, 1)$. We expect the resulting parametrisation $(x_0, y_0, a, b, h_1, h_2, r_0)$ to be numerically stable for a capsule whose centroid lies close to the xy-plane, and whose axis is roughly parallel to the z-axis.

Our second requirement is to supply a module, **cap_dgd.m**, for evaluating the distance to a capsule parametrised as above, as well as the derivatives of the distance with respect to these parameters. The specification of such a module is given below.

cap_dgd.m

```
  function [d, gradd] = cap_dgd(X, aa, NX)
%
% Distance function and gradient calculation for capsule composed
% of a cylinder with attached hemi-spherical caps. The capsule is
% parametrized by (x0, y0, a, b, h1, h2, r0).
%
% Input    X  - m x 3 array of x, y and z coordinates.
%         aa  - 7 x 1 array of parameters (x0, y0, a, b, h1, h2, r0)'.
%         NX  - 3 x 1 array defining the partitioning of X: NX(1) is
%                  the number of points that relate to the cylindrical
%                  part of the capsule, and NX(2) and NX(3) are the
%                  number of points relating to the two caps.
%
% Output   d  - m x 1 array of distances.
%      gradd  - m x 7 array of gradients.
%
%  [d, gradd] = cap_dgd(X, aa, NX)
```

In a similar way to the minimum zone circle problem, we can use **cap_dgd.m** in conjunction with **zotcon.m** to evaluate the required constraint functions and their gradients for the optimisation.

```
[c, gradc] = zotcon(X, [s; aa], 'cap_dgd', NX);
```

To construct **cap_dgd.m**, we suppose that we have available the modules **cy_dgd.m** and **sp_dgd.m** that provide, respectively, the distance to a cylinder defined by (x_0, y_0, a, b, r_0) and to a sphere defined by its centre coordinates and radius. These are more natural representations for the individual components than that provided by the parameters that define the capsule. Further, **cy_dgd.m** and **sp_dgd.m** calculate the derivatives of the distance with respect to their natural parameters.

cy_dgd.m

```
   function [d, gradd] = cy_dgd(X, aa)
%
%  Distance function and gradient evaluation for a cylinder
%  parametrized by (x0, y0, a, b, r0)'.
%
%  Input    X  - m x 3 array of x, y and z coordinates.
%           aa  - 5 x 1 array of parameters (x0, y0, a, b, r0)'.
%
%  Output   d  - m x 1 array of distances.
%       gradd  - m x 5 array of gradients.
```

sp_dgd.m

```
   function [d, gradd] = sp_dgd(X, aa)
%
%  Distance function and gradient evaluation for a sphere
%  parametrized by centre coordinates (c1, c2, c3) and radius r0.
%
%  Input    X  - m x 3 array of x, y and z coordinates.
%           aa - 4 x 1 array of parameters (c1, c2, c3, r0)'.
%
%  Output   d  - m x 1 array of distances.
%        gradd - m x 4 array of gradients.
```

To compute the required derivative values from the information returned by these element specific modules, we proceed as follows. Suppose, for a given element, the equation

$$\mathbf{b} = \mathbf{p}(\mathbf{a})$$

describes how the parameters **a** for the multi-component artefact are related to the parameters **b** for the element. Then, using the chain rule for differentiation, we have

$$\nabla_a d = J_p \nabla_b,$$

where the (i, j)-th element of J_p is $\partial p_j / \partial a_i$. Performing the calculations in this way enables us to take a *modular* approach to writing the function **cap_dgd.m**, and allows re-usability of the modules **cy_dgd.m** and **sp_dgd.m** within other applications.

To illustrate the application of this software we consider a set of data obtained by randomly perturbing points sampled from the surface of a given capsule. We use the parameter values that define the underlying artefact as the initial estimates for the optimisation process. These values, together with the corresponding Chebyshev error s, are listed in Table 3. Also listed in this table are the values found by the optimisation software that define the minimum zone capsule for the data.

Table 3 Details of the initial estimate and solution to the minimum zone problem for a capsule. Number of major iterations to find the solution is six.

	Initial estimate	Solution
x_0	-0.209144	-0.280296
y_0	0.562148	0.595564
a	0.0	-0.000270
b	0.100335	0.099437
h_1	1.990008	2.088034
h_2	-1.990008	-1.932141
r_0	0.5	0.554337
s	0.234594	0.133573

Example: minimum zone ellipse. To illustrate Chebyshev form assessment for more general curves and surfaces we consider the problem of finding the minimum zone ellipse for given data in 2D. Unlike the simpler geometric elements already considered, it is not possible to derive an explicit formula for the distance of a general point to an ellipse and we have to introduce auxiliary parameters **u** as described in section 4.

We can regard an ellipse as a parametric curve specified in terms of parameters (a, b, U, V, S) by

$$\begin{aligned} x(u) &= a + r(u)\cos u, \\ y(u) &= b + r(u)\sin u, \end{aligned}$$

where

$$r(u) = \frac{S}{\sqrt{[1 - U\cos 2u - V\sin 2u]}}$$

and U and V satisfy $U^2 + V^2 \leq 1$; see [15]. As in previous examples, to solve the minimum zone ellipse problem, we must supply a module for evaluating the distance to an ellipse specified by (a, b, U, V, S) as well as the derivatives of the distance with respect to these parameters. A Matlab module **el_dgd.m**, the header for which is listed below, provides this information.

el_dgd.m

```
   function [d, gradd] = el_dgd(X, aa)
%
%  Distance function and gradient evaluation for an ellipse in the plane
%  parametrized by (a, b, U, V, S)'.
%
%  Input    X  - m x 2 array of x and y coordinates.
%          aa  - 5 x 1 array of parameters (a, b, U, V, S)'.
%
%  Output   d  - m x 1 array of distances.
%       gradd  - m x 5 array of gradients.
%
%  Global variable
%    ellipseu  - m x 1 array of auxiliary parameters. On entry, ellipseu
%                    contains initial estimates of these parameters;
%                    on exit, ellipseu contains updated values.
```

Given the measured point (x_i, y_i) and values for the ellipse parameters (a, b, U, V, S), the module implements Newton's method for finding the point $(x(u_i^*), y(u_i^*))$ on the ellipse closest to (x_i, y_i). The distance from (x_i, y_i) to the ellipse is then given by the (signed) distance from (x_i, y_i) to $(x(u_i^*), y(u_i^*))$, and the derivatives may be calculated using (16). In order to apply Newton's method it is necessary to supply initial estimates for the values u_i^* that define the closest points. Appropriate estimates are provided by the values of these parameters computed during the previous call to the module. In order to make these values available without interfering with the form of the call to **el_dgd.m** we have assumed that these parameters are stored in the *global* Matlab variable `ellipseu`. For implementations of this module in other computing languages and environments, it may be necessary to use alternative mechanisms for dealing with these parameters: for example, in FORTRAN we may use the facility of variables in COMMON.

In Figure 2 we show a set of data obtained by randomly perturbing points sampled from the ellipse defined by parameter values $a = 0$, $b = 0$, $U = 0.5$, $V = 0.1$ and $S = 1$. We also

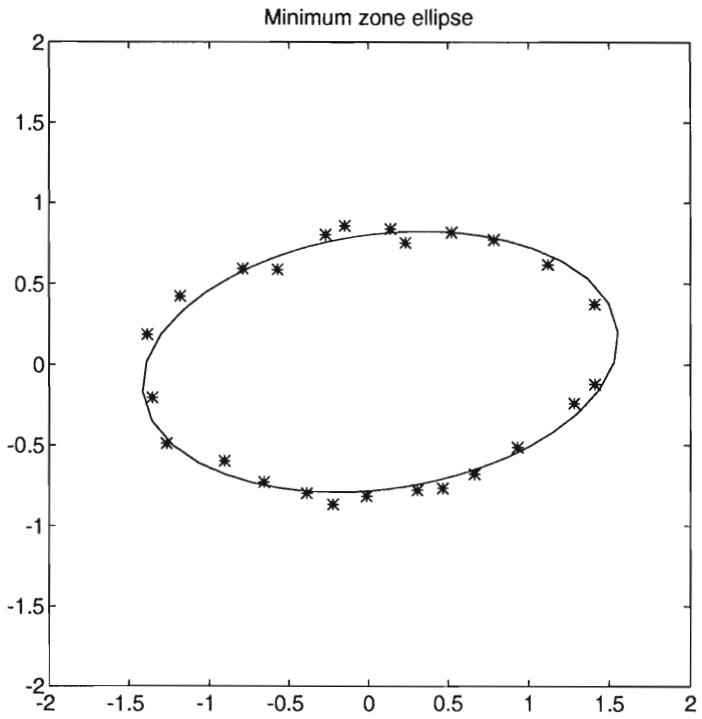


Figure 2 Minimum zone ellipse for given data.

illustrate the minimum zone ellipse for the data. In Table 4 we list the parameter values for an initial estimate of the solution ellipse, obtained from an algorithm for finding the "linear" least squares ellipse [15], together with the parameter values for the minimum zone ellipse. We indicate also the Chebyshev error s associated with this initial estimate and the solution.

Table 4 Details of the initial estimate and solution to the minimum zone ellipse problem. Number of major iterations to find the solution is four.

	Initial estimate	Solution
a	0.028139	0.069222
b	0.006233	0.016466
U	0.504038	0.543422
V	0.109834	0.154791
S	0.998841	0.986294
s	0.108912	0.076947

5.1 BCR CHEBYSHEV REFERENCE SOFTWARE PROJECT

The problem of finding minimum zone, maximum inscribed and minimum circumscribed best-fit geometric elements to data recorded by a coordinate measurement system is the subject of [1, 2, 19]. These reports summarise work carried out under a European Communities' Bureau of Reference (BCR) contract that led to the development of Chebyshev reference software for the assessment of geometric form. The geometric elements considered in the project are the line, plane, circle, sphere, cylinder and cone.

We indicated earlier that these element fitting problems could be posed as constrained optimisation problems. An important observation made in [1, 2, 19] is that the optimisation problems for the line (2D), plane, circle (2D) and sphere can be formulated as *linearly* constrained, whereas those for the line (3D), circle (3D), cylinder and cone involve *nonlinear* constraints. In implementing a feasible descent algorithm (see section 2.2) for these problems, this distinction is explicitly exploited [19]. Moreover, for the linearly constrained problems an alternative algorithm is presented [1, 2], based on a combinatorial approach, that returns all *global* solutions to these problems.

A second important observation is that these problems admit solutions for which the Kuhn-Tucker equations (see section 2.1) are rank deficient; for example, if the number of constraints active at the solution exceeds the number of parameters needed to define the geometric element. In the implementation of a feasible descent algorithm, it is necessary to deal properly with the situation where these equations become rank deficient or "close" to rank deficient. The reference software developed during the project implemented comprehensive methods for dealing with degeneracy problems.

5.2 LEAST SQUARES FORM ASSESSMENT

The zone tolerance assessment problem is a Chebyshev approximation problem for which there exists a least squares (Gaussian) counterpart. Algorithms for least squares form assessment are considered in [4, 16, 17].

5.3 LINEARISED PROBLEMS

Least squares form assessment is, in general, a much simpler problem compared with Chebyshev form asessment. Further, or alternative, simplifications may be made by considering *linearised* problems. For example, for the Chebyshev line problem (2D) suppose we measure deviations in a fixed direction (parallel to the y-axis, say) rather than orthogonal to the line itself. In this case, the distance function is a linear function of the line parameters, and the "linear Chebyshev line" that solves the associated Chebyshev problem is found using, for example, the algorithm in [5]. If the data is rotated so that the line is approximately horizontal, the linear and true Chebyshev lines may be identical. A condition for this equivalence can be expressed in terms of the minimum separation in the x-direction of the data and the (estimated) Chebyshev separation.

In [16] linearised versions of least squares form assessment problems are presented for certain of the geometric elements. The solutions to these problems are useful in providing inexpensive starting estimates for applying the algorithms discussed in this report.

5.4 NOTES AND REFERENCES

Chebyshev approximation problems have received much attention, including [10, 25, 28, 30]. In particular, successful algorithms for solving linear Chebyshev approximation problems are available [5]: see also [9]. Tolerancing for circles is set out in BS3730 [7] and algorithms for the related optimisation problems are described in [4].

6 TEMPLATE MATCHING

In this section we consider algorithms for solving the template matching problem. As explained in [18, section 6] template matching is one of the simplest types of tolerance assessment problems since the only parameters involved are those defining a change in the frame of reference. It is also common in practice.

Suppose as before that $\mathbf{a} \mapsto \mathcal{S}(\mathbf{a})$ represents the nominal form. Given ideal parameters $\mathbf{a}_0$, $\mathcal{S}(\mathbf{a}_0)$ specifies a fixed surface or *template* in a fixed frame of reference. Suppose also that data points $X = \{\mathbf{x}_i : i \in I\}$ are gathered in the surface of a workpiece by a CMS. The coordinates of the data points $\mathbf{x}_i$ will be in a different frame of reference from that of $\mathcal{S}(\mathbf{a}_0)$. The purpose of template matching is to see to what extent the measured workpiece differs from the specified template. This is achieved by finding a frame transformation T which maps the data onto the template in such a way that the transformed data points satisfy certain form constraints.

We suppose that the transformation T is determined by parameters $\mathbf{t}$ and T transforms $\mathbf{x}$ to $\hat{\mathbf{x}}$:

$$\hat{\mathbf{x}} = T(\mathbf{t}; \mathbf{x}).$$

If the form constraints are $\phi_i(X, \mathbf{a}) \geq 0$, $i \in \Phi$, then the template matching problem is stated as:

Given $X = \{\mathbf{x}_i : i \in I\}$ and the template $\mathcal{S}_0 = \mathcal{S}(\mathbf{a}_0)$, find transformation parameters $\mathbf{t}$ such that

$$\phi_i(\hat{X}, \mathbf{a}_0) \geq 0, \quad i \in \Phi,$$

where $\hat{X} = \{\hat{\mathbf{x}}_i : \hat{\mathbf{x}}_i = T(\mathbf{t}; \mathbf{x}_i),\ i \in I\}$.

Notice that $\phi_i(\hat{X}, \mathbf{a}_0)$ depends on the transformation parameters $\mathbf{t}$, so we write ϕ_i as $\phi_i(\mathbf{t}, \mathbf{a}_0)$. The associated optimisation problem is then

$$\min_{s,\mathbf{t}} s \quad \text{subject to} \quad s + \phi(\mathbf{t}, \mathbf{a}_0) \geq 0, \quad i \in \Phi. \tag{28}$$

To use optimisation software to solve this problem, we need a module to calculate ϕ_i and $\nabla_{\mathbf{t}}\phi_i$.

In most template matching problems the form constraints are a combination of zone, inscribing and circumscribing constraints involving the distance $d(\hat{\mathbf{x}}; \mathbf{a}_0)$ from a point $\hat{\mathbf{x}}$ to the profile $\mathcal{S}_0$. Consequently, we usually have

$$\phi_i(\mathbf{t}; \mathbf{a}_0) = \pm d(\hat{\mathbf{x}}_i, \mathbf{a}_0).$$

6.1 TEMPLATE MATCHING IN TWO DIMENSIONS

In this section we suppose that $\mathcal{S}(\mathbf{a}_0)$ is a fixed profile in the xy-plane. (We use the term *profile* for a template in two dimensions.)

A general affine transformation T in two dimensions is expressed in terms of three parameters $\mathbf{t} = (x_0, y_0, \theta)^{\mathrm{T}}$ by

$$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix}. \tag{29}$$

We note that

$$\begin{aligned} \frac{\partial}{\partial x_0}(\hat{x}, \hat{y}) &= (-\cos\theta, \sin\theta), \\ \frac{\partial}{\partial y_0}(\hat{x}, \hat{y}) &= (-\sin\theta, -\cos\theta), \\ \frac{\partial}{\partial \theta}(\hat{x}, \hat{y}) &= (\hat{y}, -\hat{x}), \end{aligned} \tag{30}$$

which together with (21) show how $\nabla_{\mathbf{t}} d(\hat{\mathbf{x}}(\mathbf{t}), \mathbf{a})$ can be calculated. In detail,

$$\begin{aligned} \frac{\partial d}{\partial x_0} &= -n_x \cos\theta + n_y \sin\theta, \\ \frac{\partial d}{\partial y_0} &= -n_x \sin\theta - n_y \cos\theta, \\ \frac{\partial d}{\partial x_0} &= n_x \hat{y} - n_y \hat{x}, \end{aligned}$$

where $\mathbf{n}^* = (n_x, n_y)$ is the normal at $\mathbf{x}^*(\hat{\mathbf{x}}, \mathbf{a}_0)$, the closest point on the profile to $\hat{\mathbf{x}}$.

The Matlab function **fpmdgd.m**, specified as follows, evaluates the distance function and its gradient for the profile matching problem by implementing the formulæ given above.

fpmdgd.m

```
  function [ d, gradd, Xh, N ] = fpmdgd(X,tt,cu_dnl,p1,p2,p3,p4,p5,p6)
%
% Distance function and gradient evaluation for the profile matching
% problem.
%
% Input     X  - m x 2 array of x and y coordinates.
%          tt  - 3 x 1 array of parameters (x0, y0, theta)'.
%
%      cu_dnl  - function to evaluate distances and normals
%                specified by
%
%                  [ d, N ] = cu_dnl(X, p1, p2, ...)
%
%     p1,p2,.. - up to six additional parameters to be passed to the
%                function cu_dnl.
%
% Output    d  - m x 1 array of distances.
%       gradd  - m x 3 array of distance gradients.
%          Xh  - m x 2 array of transformed data points.
%           N  - m x 2 array of normals.
%
% [ d, gradd, Xh, N ] = fpmdgd(X,tt,'cu_dnl',<p1,p2,p3,p4,p5,p6>)
```

The module is almost completely generic in that the only profile specific component appears in the submodule **cu_dnl.m** with specification:

cu_dnl.m

```
  function [ d, N ] = cu_dnl(X, p1, p2, ...)
%
% Distance function and normal vector evaluation for a fixed profile.
%
%  Input    X  - m x 2 array of x and y coordinates (xi, yi).
%
%    p1,p2,..  -       additional parameters.
%
% Output    d  - m x 1 array of distances from (xi, yi) to the profile.
%           N  - m x 2 array containing the normals to the profile
%                      corresponding to the points X.
```

Moreover, **fpmdgd.m** can be used in conjunction with **zotcon.m**, for example, to solve the profile matching problem with zone tolerances. This is illustrated below.

Example: n **circles.** Consider a fixed profile determined by n_K circles $\mathcal{C}_k = \mathcal{C}(\mathbf{a}_k) = \mathcal{C}(a_k, b_k, r_k)$ centred at points (a_k, b_k) with radii r_k, $k = 1, \ldots, n_K$. The tolerance assessment problem we address is: given sets of measured data $X_k = \{\mathbf{x}_i : i \in I_k\}$, find an affine transformation $\mathbf{t} = (x_0, y_0, \theta)^{\mathrm{T}}$ of the data such that the transformed points $\hat{X}_k$ lie at most a distance τ from $\mathcal{C}_k$.

The corresponding optimisation problem is

$$\min_{s,\mathbf{t}} s$$

subject to the constraints

$$\begin{aligned} s + d(\hat{\mathbf{x}}_i(\mathbf{t}),\mathcal{C}_k) &\geq 0, \quad i \in I_k, \quad k = 1, \ldots, n_K, \\ s - d(\hat{\mathbf{x}}_i(\mathbf{t}),\mathcal{C}_k) &\geq 0, \quad i \in I_k, \quad k = 1, \ldots, n_K. \end{aligned}$$

If at the solution $s \leq \tau$, the tolerance constraints are satisfied.

To use the module **fpmdgd.m** we supply a module **nc_dnl.m** to calculate the distances $d(\hat{\mathbf{x}}_i, \mathbf{a}_k)$ and normals $\mathbf{n}(\hat{\mathbf{x}}_i^*, \mathbf{a}_k)$, $i \in I_k$, associated with the sets of data points X_k and circles $\mathcal{C}_k$, $k = 1, \ldots, n_K$.

nc_dnl.m

```
   function [d, N] = nc_dnl(X, aa, NX)
%
%  Distance function and normal vector evaluation for n circles.
%
%  Input    X  - m x 2 array of x and y coordinates.
%          aa  - n x 3 array containing the circle parameters:
%                    aa(k, 1:2) contains the centre coordinates
%                    for the kth circle, and aa(k, 3) is its radius.
%          NX  - n x 1 array defining the partitioning of X:
%                    NX(k) is the number of points associated
%                    with the kth circle.
%
%  Output   d  - m x 1 array of distances.
%           N  - m x 2 array of normal vectors.
```

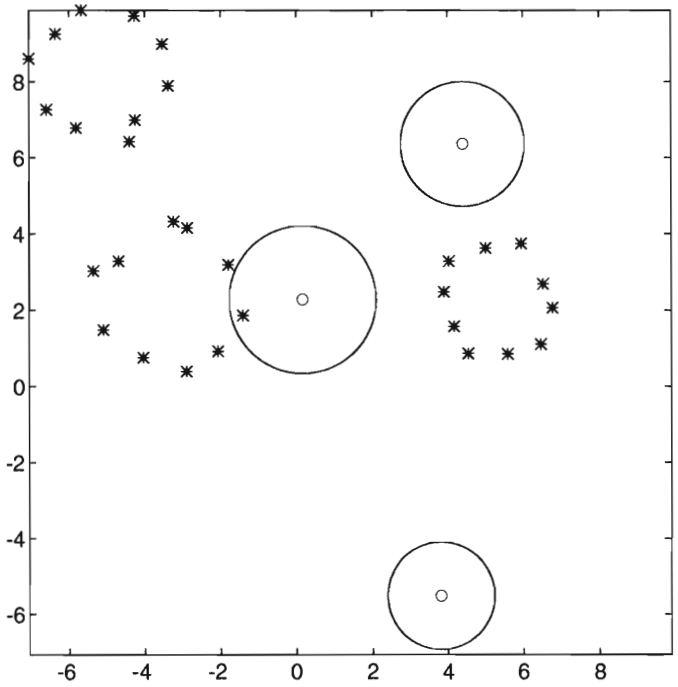


Figure 3 Specified profile and measured data for the n circles problem.

fpmdgd.m can in turn be supplied to the module **zotcon.m** which assembles the constraint functions and gradients for zone tolerance assessment through a module call of the form

```
[c, gradc] = zotcon(X, [s; tt] , 'fpmdgd', 'nc_dnl', aa, NX);
```

In Figure 3 we illustrate a fixed profile composed of three circles together with data X_k, $k \in \{1,2,3\}$, that nominally represent the profile. The data is obtained by randomly perturbing points sampled from the profile, and then applying an affine transformation to these points.

Initial estimates of the affine transformation that maps the data onto the profile may be found in the following way: we compute the centroids of the data sets X_1 and X_2 that represent the first two circles, and determine the transformation that maps the line segment joining these centroids to the line segment joining the centres of $\mathcal{C}_1$ and $\mathcal{C}_2$. In Table 5 we list initial estimates of $\mathbf{t} = (x_0, y_0, \theta)^{\mathrm{T}}$ for the data and profile shown in Figure 3.

Figure 4 shows the solution to the n-circles problem for this data and Table 5 gives the values of the transformation parameters that define the solution. We also indicate in this table the Chebyshev error s for the initial estimate of $\mathbf{t}$ and at the solution. s is the maximum departure (in absolute value) of the transformed data points from the profile.

In principle, *any* zone tolerance assessment problem for profile matching can be addressed using the harness modules **zotcon.m** and **fpmdgd.m** by supplying an appropriate module in place of **cu_dnl.m** to calculate distances and normal vectors.

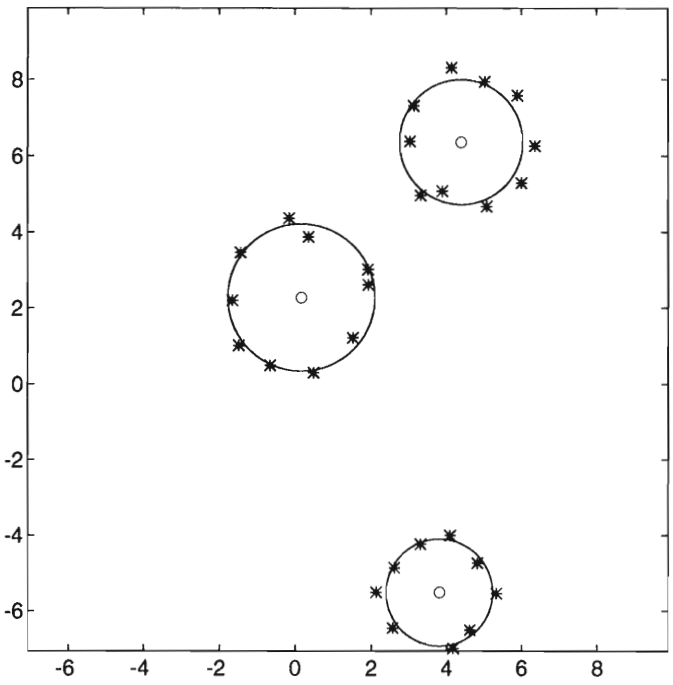


Figure 4 Solution to the n circles problem for the profile and data of Figure 3.

Table 5 Details of the initial estimate and solution to the n-circles problem. Number of major iterations to find the solution is four.

	Initial estimate	Solution
x_0	-1.455126	-1.345391
y_0	1.380721	1.299318
θ	1.101539	1.127410
s	0.461938	0.325150

6.2 TEMPLATE MATCHING IN THREE DIMENSIONS

The most general affine transformation $\hat{\mathbf{x}} = T(\mathbf{t};\mathbf{x})$ in three dimensions can be expressed as

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} = R_3R_2R_1U_0 \begin{bmatrix} x - x_0 \\ y - y_0 \\ z - z_0 \end{bmatrix}. \tag{31}$$

Here,

$$R_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_1 & s_1 \\ 0 & -s_1 & c_1 \end{bmatrix}, \; R_2 = \begin{bmatrix} c_2 & 0 & s_2 \\ 0 & 1 & 0 \\ -s_2 & 0 & c_2 \end{bmatrix} \text{ and } R_3 = \begin{bmatrix} c_3 & s_3 & 0 \\ -s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

are plane rotation matrices specified by $c_k = \cos\theta_k$ and $s_k = \sin\theta_k$, $k = 1,2,3$, defining rotations about the x-, y- and z-axes, respectively, and U_0 is a fixed rotation matrix.

The use of the most general affine transformation will not be appropriate if the template has rotational symmetry or translational symmetry. For example, if the template is a cylinder whose axis is the z-axis then a suitable transformation can be constructed by holding θ_3 and z_0 constant. For this form of rotational symmetry, the matrix U_0 is used to rotate the axis of the measured cylinder so that it is approximately parallel to the z-axis and, in this way, ensure the numerical stability of the parametrization chosen for the underlying nominal form. For a fuller discussion on the stability of the parametrization of geometric form, see [3].

Derivatives of $\hat{\mathbf{x}}$ with respect to $\mathbf{t}$ for such a transformation are:

$$\begin{aligned} \frac{\partial\hat{\mathbf{x}}}{\partial x_0} &= (-1,0,0)R^{\mathrm{T}}, \\ \frac{\partial\hat{\mathbf{x}}}{\partial y_0} &= (0,-1,0)R^{\mathrm{T}}, \\ \frac{\partial\hat{\mathbf{x}}}{\partial z_0} &= (0,0,-1)R^{\mathrm{T}}, \\ \frac{\partial\hat{\mathbf{x}}}{\partial\theta_k} &= (x - x_0, y - y_0, z - z_0)R_{\theta_k}^{\mathrm{T}}, \end{aligned}$$

where $R = R_3R_2R_1U_0$, and $R_{\theta_k} = \frac{\partial R}{\partial\theta_k}$, $k = 1,2,3$, can be calculated according to

$$\begin{aligned} R_{\theta_1} &= R_3R_2\dot{R}_1U_0, \\ R_{\theta_2} &= R_3\dot{R}_2R_1U_0, \\ R_{\theta_3} &= \dot{R}_3R_2R_1U_0, \end{aligned}$$

with

$$\dot{R}_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -s_1 & c_1 \\ 0 & -c_1 & -s_1 \end{bmatrix}, \; \dot{R}_2 = \begin{bmatrix} -s_2 & 0 & c_2 \\ 0 & 0 & 0 \\ -c_2 & 0 & -s_2 \end{bmatrix} \text{ and } \dot{R}_3 = \begin{bmatrix} -s_3 & c_3 & 0 \\ -c_3 & -s_3 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The Matlab function specified below gives the three dimensional version of Module **fpmdgd.m**. The parameter `ttL` is used to select which derivatives are to be calculated so that the same module can be used for transformations appropriate for templates with different symmetries.

ftmdgd.m

```
 function [ d, gradd, Xh, N ] = ftmdgd(X,tt,su_dnl,ttL,p1,p2,p3,p4,p5,p6)
%
% Distance function and gradient evaluation for the template matching
% problem.
%
% Input     X  - m x 3 array of x, y and z coordinates.
%          tt  - t x 1 array of parameters, a subset of
%                      (x0, y0, z0, theta1, theta2, theta3)'.
%
%      su_dnl  - function to evaluate distances and normals
%                specified by
%
%                  [ d, N ] = su_dnl(X, p1, p2, ...)
%
%         ttL  - 6 x 1 array indicating which transformation
%                      parameters are active.
%
%     p1,p2,.. - up to six additional parameters to be passed to
%                the function su_dnl.
%
% Output    d  - m x 1 array of distances.
%       gradd  - m x t array of distance gradients, where t is the
%                      number of active transformation parameters.
%           Xh - m x 3 array of transformed data points.
%            N - m x 3 array of normals.
%
%  [ d, gradd, Xh, N ] = ftmdgd(X,tt,'su_dnl',ttL,<p1,p2,p3,p4,p5,p6>)
```

Again, the surface specific part of the module is restricted to the module **su_dnl.m** specified by:

su_dnl.m

```
 function [ d, N ] = su_dnl(X, p1, p2, ...)
%
% Distance function and normal vector evaluation for a fixed template.
%
% Input     X  - m x 3 array of x, y and z coordinates.
%
%    p1,p2,..  -       additional parameters.
%
% Output    d  - m x 1 array of distances from (xi, yi, zi) to
%                      the template.
%           N  - m x 3 array containing the normals to the template
%                      corresponding to the points X.
```

Example: n spheres. Suppose the fixed template is made up of n_K spheres $\mathcal{S}_k$ centred at $\mathbf{a}_k$ with radii r_k and that the tolerance assessment problem is:

Given sets of measurements $X_k = \{\mathbf{x}_i : i \in I_k\}$, find an affine transformation of the data such that the transformed data points $\hat{X}_k$ lie at most distance τ from S_k.

The corresponding optimisation problem is

$$\min_{s,\mathbf{t}} s$$

subject to the constraints

$$\begin{aligned} s + d_k(\hat{\mathbf{x}}_i(\mathbf{t}), \mathcal{S}_k) &\geq 0, \quad i \in I_k, \quad k = 1, \ldots, n_K, \\ s - d_k(\hat{\mathbf{x}}_i(\mathbf{t}), \mathcal{S}_k) &\geq 0, \quad i \in I_k, \quad k = 1, \ldots, n_K. \end{aligned}$$

If at the solution $s \leq \tau$ then the tolerance constraints can be satisfied.

To solve this problem we require the distance evaluation module:

ns_dnl.m

```
   function [d, N] = ns_dnl(X, aa, NX)
%
%  Distance function and normal vector calculations for n spheres.
%
%  Input    X  - m x 3 array of x, y and z coordinates.
%           aa  - n x 4 array containing the sphere parameters
%                  for the template: aa(k, 1:3) contains the
%                  centre coordinates for the kth sphere, and
%                  aa(k, 4) is its radius.
%           NX  - n x 1 array defining the partitioning of X:
%                  NX(k) is the number of points associated
%                  with the kth sphere.
%
%  Output   d  - m x 1 array of distances.
```

ns_dnl.m is an instantiation of **su_dnl.m** that may be supplied as input to **ftmdgd.m** for calculating distances to this particular template as well as the derivatives of these distances with respect to the transformation parameters $\mathbf{t}$. These modules can be used in conjunction with **zotcon.m**.

For this example we consider a fixed template composed of three spheres. Data is obtained by applying an affine transformation to points sampled from these spheres. The transformation that maps the data to the template is defined by

$$\begin{aligned} x_0 &= -0.884457, \\ y_0 &= -1.215551, \\ z_0 &= -0.008884, \\ \theta_1 &= -0.313805, \\ \theta_2 &= 0.909032, \\ \theta_3 &= 0.491255. \end{aligned}$$

The data points are also subject to random perturbations, and so we expect the optimisation software to return values for the transformation parameters that are "close" to those listed above.

Table 6 lists initial estimates for the transformation parameters together with the values of these parameters at the computed solution. The table also gives the Chebyshev error s for the initial estimates and at the solution. s is the maximum departure (in absolute value) from the transformed data points to the fixed template.

Table 6 Details of the initial estimate and solution to the n-spheres problem. Number of major iterations to find the solution is eleven.

	Initial estimate	Solution
x_0	-0.910932	-0.882484
y_0	-1.166797	-1.201984
z_0	0.036872	0.029980
θ_1	-0.337653	-0.314544
θ_2	0.784125	0.914115
θ_3	0.631958	0.490765
s	2.484568	0.058147

Example: n parallel cylinders. Consider a fixed template determined by n_K cylinders $\mathcal{C}_k$ with radii r_k, $k = 1, \ldots, n_K$, whose axes are parallel to the z-axis and pass through the points $(x_k, y_k, 0)$, $k = 1, \ldots, n_K$. The tolerance assessment problem we address is: given sets of measured data $X_k = \{\mathbf{x}_i : i \in I_k\}$, find an affine transformation of the data such that the transformed points $\hat{X}_k$ lie inside the kth cylinder.

The template has translational symmetry along the z-axis, and so we hold z_0 constant in the transformation T. Thus, T depends on the *five* parameters $\mathbf{t} = (x_0, y_0, \theta_1, \theta_2, \theta_3)^{\mathrm{T}}$. The corresponding optimisation problem is

$$\min_{s,\mathbf{t}} s$$

subject to the constraints

$$s - d(\hat{\mathbf{x}}_i(\mathbf{t}), \mathcal{C}_k) \geq 0, \quad i \in I_k.$$

If at the solution $s \leq 0$, the tolerance constraints are satisfied.

We can use **ftmdgd.m** in conjunction with a module **pc_dnl.m** which calculates the appropriate distances and normal vectors. Notice that when using **ftmdgd.m** to solve this problem, we assign `ttL` to be $(1, 1, 0, 1, 1, 1)^{\mathrm{T}}$ to reflect the fact that z_0 is constant.

pc_dnl.m

```
   function [d, N] = pc_dnl(X, aa, NX)
%
%  Distance function and normal vector evaluation for n parallel cylinders.
%
%  Input    X  - m x 3 array of x, y and z coordinates.
%          aa  - n x 3 array containing the cylinder parameters
%                    for the template: aa(k, 1:2) contains the
%                    coordinates (ak, bk) of the locating point
%                    for the axis of the kth cylinder, and aa(k, 3)
%                    is its radius.
%          NX  - n x 1 array defining the partitioning of X:
%                    NX(k) is the number of points associated
%                    with the kth cylinder.
%
%  Output   d  - m x 1 array of distances.
%           N  - m x 3 array of normal vectors.
```

Table 7 Details of the initial estimate and solution to the n-parallel cylinders problem. Number of major iterations to find the solution is five.

	Initial estimate	Solution
x_0	0.463054	0.299474
y_0	1.216710	1.284401
θ_1	-0.097561	-0.249354
θ_2	-0.750360	-0.770896
θ_3	-0.215014	-0.214850
s	0.928137	0.010125

For this example we consider a fixed template composed of three cylinders with axes that are parallel to the z-axis. Data is obtained by applying an affine transformation to points sampled from these cylinders. The transformation that maps the data to the template is defined by

$$\begin{aligned} x_0 &= 0.302722, \\ y_0 &= 1.288666, \\ \theta_1 &= -0.250085, \\ \theta_2 &= -0.771591, \\ \theta_3 &= -0.214823. \end{aligned}$$

The data points are also subject to random perturbations, and so we expect the optimisation problem we solve to return values for the transformation parameters that are "close" to those listed above.

Table 7 lists initial estimates for the transformation parameters together with the values of these parameters at the computed solution. The table also gives the value of s for the initial estimates and at the solution. Since $s > 0$ at the solution, we conclude that there exists no transformation for which the transformed data points lie inside the cylinders that define the template.

6.3 LEAST SQUARES TEMPLATE MATCHING

The template matching problem with zone tolerances is a Chebyshev approximation problem for which there exists a least squares (Gaussian) counterpart:

Given a fixed template $\mathcal{S}_0$ and a data set $X = \{\mathbf{x}_i : i \in I\}$,

$$\min_{\mathbf{t}} \sum_{i \in I} d^2(\hat{\mathbf{x}}_i, \mathcal{S}_0),$$

where, as before, $\hat{\mathbf{x}} = T(\mathbf{t}; \mathbf{x})$. Since the modules **fpmdgd.m** and **ftmdgd.m** calculate d and $\nabla_{\mathbf{t}} d$, they may also be used to supply function and gradient information to least squares optimisation software.

The least squares will in general be different from the Chebyshev solution. Nevertheless, the least squares formulation can give a solution that shows that the tolerances are met, even though the solution is not optimal in the Chebyshev sense.

7 PART MATING

One of the most important tolerance assessment problems is to decide whether two parts S_X and S_Y will mate and, if so, how closely they fit together. We assume that the surfaces of the parts S_X and S_Y each have nominal form $\mathbf{a} \mapsto \mathcal{S}(\mathbf{a})$ and we try to find a *separating surface* $\mathcal{S}(\mathbf{a})$ such that S_X lies on one side of $\mathcal{S}(\mathbf{a})$ and S_Y lies on the other. This information can be encoded using *separation constraints* of the form

$$d(\mathbf{x}; \mathcal{S}(\mathbf{a})) \;\geq\; 0 \;\geq\; d(\mathbf{y}; \mathcal{S}(\mathbf{a})),$$

for $\mathbf{x} \in S_X$ and $\mathbf{y} \in S_Y$. In practice, measurements $X = \{\mathbf{x}_i : i \in I_X\}$ and $Y = \{\mathbf{y}_i : i \in I_Y\}$ of points on the surfaces S_X and S_Y will be made separately so that the sets X and Y will not, in general, be in the same frame of reference. Thus, to solve the tolerance assessment problem we have to introduce a frame transformation $\hat{\mathbf{y}} = T(\mathbf{t}; \mathbf{y})$ to transform the data points in Y to the frame of reference of X.

The problem of finding a separating surface can then be posed as

$$\min_{s,\mathbf{t},\mathbf{a}} s \quad \text{subject to} \quad \begin{cases} s + d(\mathbf{x}_i, \mathbf{a}) \;\geq\; 0, & i \in I_X, \\ s - d(\hat{\mathbf{y}}_i, \mathbf{a}) \;\geq\; 0, & i \in I_Y, \end{cases} \tag{32}$$

where $\hat{\mathbf{y}}_i = T(\mathbf{t}; \mathbf{y}_i)$. If at the solution to this problem $s \leq 0$, we may conclude that a separating surface exists and the two surfaces S_X and S_Y mate.

The constraints involving the data points X depend only on the parameters $\mathbf{a}$, whereas those involving the data points Y depend on $\mathbf{a}$ and the transformation parameters $\mathbf{t}$. Thus, the constraint gradients will involve $\partial d(\mathbf{x}, \mathbf{a})/\partial a_j$, $\partial d(\hat{\mathbf{y}}, \mathbf{a})/\partial a_j$ and $\partial d(\hat{\mathbf{y}}, \mathbf{a})/\partial t_j$. These can be supplied by two calls to a distance function and gradient module **ge_dgd.m** that is specific to S, and a call to a template matching module **fpmdgd.m** or **ftmdgd.m** that in turn calls **ge_dnl.m**, also specific to S. There may also be additional tolerance constraints on the surface parameters $\mathbf{a}$.

The module **ppmcon.m** specified below can be used to assemble the constraint functions and gradients for a typical multi-component part mating problem where we seek a separating *curve* in two dimensions. It is a simple matter to adapt this routine to cope with problems where we wish to determine a separating *surface* in three dimensions. The main difference between the routines is that for the problem in three dimensions we use **ftmdgd.m** instead of **fpmdgd.m**.

ppmcon.m

```
  function [ c, gradc ] = ppmcon(XY,bb,ge_dgd,ge_dnl,NX,NY,p1,p2,p3,p4)
%
% Part mating constraint function and gradient evaluation in two dimensions.
%
%     d(x, S(aa))  >=  0  >=  d(yh, S(aa)).
%
% Input
%         XY  - (mX+mY) x 2 array of points on inner and outer curves.
%         bb  -  n x 1 array of optimisation parameters (s, tt, aa)' where
%                       tt = (x0, y0, theta)'.
%     ge_dgd  -         module to calculate distance functions and
%                       gradients:
%
```

```
%                          [d, gradd] = ge_dgd(X,aa,NX,p1,p2,...).
%
%     ge_dnl  -        module to calculate distance functions and
%                      normals:
%
%                          [d, N] = ge_dnl(X,aa,NX,p1,p2,...).
%
%         NX  - nK x 1 array partitioning the X data. sum(NX) = mX.
%         NY  - nK x 1 array partitioning the Y data. sum(NY) = mY.
%
%  p1,p2,...  -        up to four additional parameters to be passed
%                      to the modules ge_dgd and ge_dnl.
%
% Output
%          c  - (mX+mY) x 1 array of constraint values.
%      gradc  - (mX+mY) x n array containing the gradients of c.
%
% Modular structure    fpmdgd
%
% [ c, gradc ] = ppmcon(XY,bb,'ge_dgd','ge_dnl',NX,NY,<p1,p2,p3,p4>)
```

Example: separating circles. In this example we suppose the two surfaces S_X and S_Y are each composed of n_K circles. S_X represents a set of circular "plugs", and S_Y a set of circular "holes". The question we ask is whether the plugs will fit inside the holes.

Let $X_k = \{\mathbf{x}_{i,k} : i \in I_k\}$, $k = 1, \ldots, n_K$, denote the measured points for the n_K holes, and $Y_k = \{\mathbf{y}_{i,k} : i \in J_k\}$, $k = 1, \ldots, n_K$, the measured points for the corresponding plugs. The separating circles problem is to find an affine transformation T with parameters $\mathbf{t}$ and a set of circles $\mathcal{C}_k = \mathcal{C}(\mathbf{a}_k) = \mathcal{C}(a_k, b_k, r_k)$, $k = 1, \ldots, n_K$, such that

$$\begin{aligned} d(\mathbf{x}_{i,k}, \mathcal{C}_k) &\geq 0, \quad i \in I_k, \quad k = 1, \ldots, n_K, \\ d(\hat{\mathbf{y}}_{i,k}, \mathcal{C}_k) &\leq 0, \quad i \in J_k, \quad k = 1, \ldots, n_K, \end{aligned}$$

where

$$\hat{\mathbf{y}}_{i,k} = T(\mathbf{t}, \mathbf{y}_{i,k}),$$

and $d(\mathbf{x}, \mathcal{C})$ is the distance from the point $\mathbf{x}$ to the circle $\mathcal{C}$. The corresponding optimisation problem is

$$\min_{\mathbf{b}} s \quad \text{subject to} \quad \left.\begin{aligned} s + d(\mathbf{x}_{i,k}, \mathcal{C}_k) &\geq 0, \quad i \in I_k, \\ s - d(\hat{\mathbf{y}}_{i,k}, \mathcal{C}_k) &\geq 0, \quad i \in J_k, \end{aligned}\right\} \quad k = 1, \ldots, n_K,$$

where $\mathbf{b}^{\mathrm{T}} = (s, \mathbf{t}^{\mathrm{T}}, \mathbf{a}_1^{\mathrm{T}}, \ldots, \mathbf{a}_{n_K}^{\mathrm{T}})$.

We can use **ppmcon.m** to evaluate the constraint functions and their derivatives with respect to the parameters **b** provided we supply modules **nc_dgd.m** and **nc_dnl.m** in place of, respectively, **ge_dgd.m** and **ge_dnl.m**. The module **nc_dnl.m** is described in section 6.1 in the context of solving the n circles profile matching problem, and we give below the specification of **nc_dgd.m**:

nc_dgd.m

```
   function [d, gradd] = nc_dgd(X, aa, NX)
%
%  Distance function and gradient evaluation for n circles.
%
%  Input    X  - m x 2 array of x and y coordinates.
%           aa - n x 3 array containing the circle parameters:
%                    aa(k, 1:2) contains the centre coordinates
%                    for the kth circle, and aa(k, 3) is its radius.
%           NX - n x 1 array defining the partitioning of X:
%                    NX(k) is the number of points associated
%                    with the kth circle.
%
%  Output   d  - m x 1 array of distances.
%        gradd - m x 3 array of gradients.
```

The call to **ppmcon.m** takes the following form:

```
[c, gradc] = ppmcon(XY, [s; aa], 'nc_dgd', 'nc_dnl', NX, NY);
```

In Figure 5 we show a set of data that represents three holes and three corresponding plugs for which, under a suitable transformation of the plug data, there exists a separating surface composed of three circles. We use the symbol "o" to denote a point measured on the surface of a hole, and the symbol "*" for a point measured on the surface of a plug. In Table 8 we list initial values for the transformation parameters $\mathbf{t}$ and the circle parameters $\mathbf{a}_k$, $k = 1, \ldots, n_K$, together with the minimax error s determined by these values. Since $s > 0$ we conclude that the initial estimates do not define a separating surface.

Also in Table 8 we list the parameter values that define the solution to the optimisation problem given above, and this solution is illustrated in Figure 6. Since $s \leq 0$ at the solution we confirm that the holes and plugs mate.

Example: separating circles with constraints on centres. We can also consider imposing constraints on the positions of the centres of the circles in the separating circles problem. If $D_{i,j}$ is the nominal value for the distance between the centres of the ith and jth circles, and $\tau_{i,j}$ is a specified tolerance on this distance, the separating circles problem with constraints on the centres takes the following form: given measured data X_k, Y_k, $k = 1, \ldots, n_K$ (as before), find an affine transformation T with parameters $\mathbf{t}$ and a set of circles $\mathcal{C}_k = \mathcal{C}(\mathbf{a}_k) = \mathcal{C}(a_k, b_k, r_k)$, $k = 1, \ldots, n_K$, such that

$$\begin{aligned} d(\mathbf{x}_{i,k}, \mathcal{C}_k) &\geq 0, \quad i \in I_k, \quad k = 1, \ldots, n_K, \\ d(\hat{\mathbf{y}}_{i,k}, \mathcal{C}_k) &\leq 0, \quad i \in J_k, \quad k = 1, \ldots, n_K, \\ |d_{i,j} - D_{i,j}| &\leq \tau_{i,j}, \quad i = 1, \ldots, j, \quad j = 1, \ldots, n_K, \end{aligned}$$

where

$$\hat{\mathbf{y}}_{i,k} = T(\mathbf{t}, \mathbf{y}_{i,k}),$$

$d(\mathbf{x}, \mathcal{C})$ is the distance from the point $\mathbf{x}$ to the circle $\mathcal{C}$, and $d_{i,j}$ is the distance between the centres (a_i, b_i) and (a_j, b_j). The corresponding optimisation problem is

$$\min_{\mathbf{b}} s \quad \text{subject to} \quad \begin{cases} s + d(\mathbf{x}_{i,k}, \mathcal{C}_k) & \geq 0, \quad i \in I_k, \quad k = 1, \ldots, n_K, \\ s - d(\hat{\mathbf{y}}_{i,k}, \mathcal{C}_k) & \geq 0, \quad i \in J_k, \quad k = 1, \ldots, n_K, \\ s - (|d_{i,j} - D_{i,j}| - \tau_{i,j}) & \geq 0, \quad i = 1, \ldots, j, \quad j = 1, \ldots, n_K, \end{cases} \tag{33}$$

Table 8 Details of the initial estimate and solution to the separating circles problem. Number of major iterations to find the solution is nine.

	Initial estimate	Solution
x_0	1.873894	2.129328
y_0	-4.970900	-4.935426
θ	0.752908	0.789805
a_1	2.002230	2.017829
b_1	5.047352	4.998370
r_1	1.196140	0.995976
a_2	6.320869	6.243932
b_2	5.038618	5.004258
r_2	1.038682	1.004928
a_3	4.870676	4.829022
b_3	2.290546	2.147494
r_3	1.248476	1.004839
s	0.167459	-0.169940

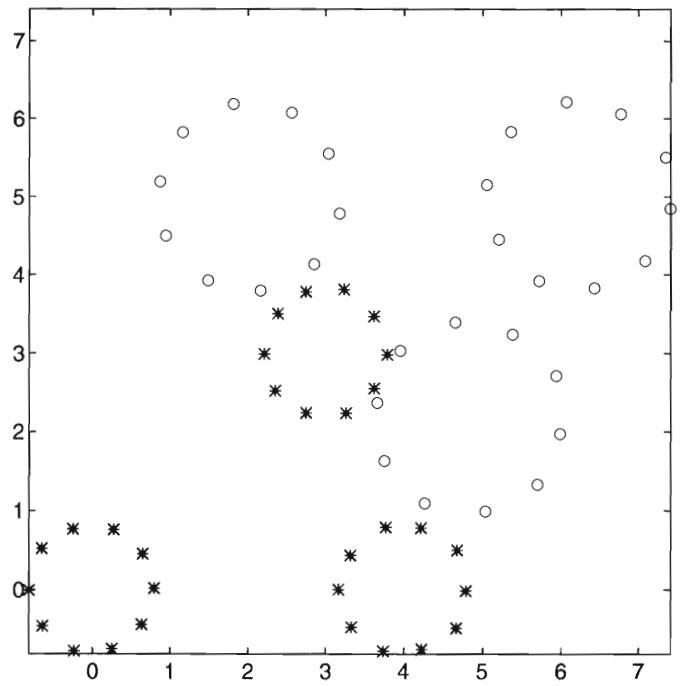


Figure 5 Data for the separating circles problem: the symbol "o" is used to represent a point measured on the surface of a hole, and the symbol "*" for a point measured on the surface of a plug.

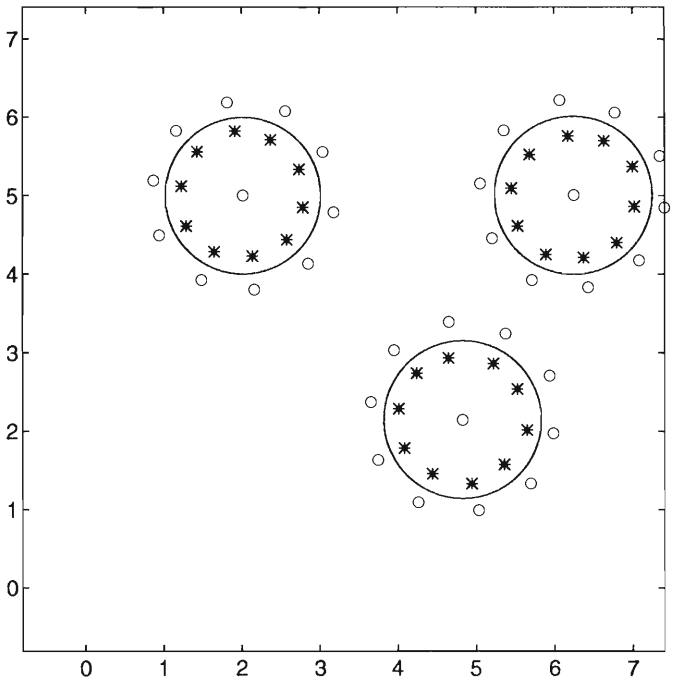


Figure 6 Solution to the separating circles problem for the data shown in Figure 5: the separating circles are indicated for the hole data and for the transformed plug data.

where $\mathbf{b}^{\mathrm{T}} = (s, \mathbf{t}^{\mathrm{T}}, \mathbf{a}_1^{\mathrm{T}}, \ldots, \mathbf{a}_{n_K}^{\mathrm{T}})$.

As in the previous example we can use the generic module **ppmcon.m** together with **nc_dgd.m** and **nc_dnl.m** to evaluate the functions relating to the form constraints together with their derivatives with respect to the parameters **b**. It remains for the user to supply a module to provide this information for the parameter constraints as well.

We consider again the data illustrated in Figure 5, except now we require that the circle centres satisfy

$$\begin{aligned} |d_{1,2} - 3.942641| &\leq 0.01, \\ |d_{2,3} - 2.862278| &\leq 0.01, \\ |d_{1,3} - 3.700000| &\leq 0.01. \end{aligned}$$

We use the parameter values that define the solution to the problem with no centre constraints (see previous example) as initial estimates for this problem. In Table 9 we list these values together with the corresponding minimax error s. Since $s > 0$, the initial estimates do not define a feasible point. Also listed in the table are the values of the parameters that define a solution to the optimisation problem, and this solution is illustrated in Figure 7. Since $s \leq 0$ at the solution, we conclude that a set of separating circles for the data that satisfy the given parameter constraints does exist.

Furthermore, the value -0.01 taken by s at the solution is significant in that it tells us that the solution circles satisfy

$$d_{i,j} = D_{i,j}, \quad i = 1, \ldots, j, \quad j = 1, \ldots, 3. \tag{34}$$

Table 9 Details of the initial estimate and solution to the separating circles problem with constraints on the circle centres. Number of major iterations to find the solution is four.

	Initial estimate	Solution
x_0	2.129328	2.128748
y_0	-4.935426	-4.929745
θ	0.789805	0.785953
a_1	2.017829	2.173840
b_1	4.998370	4.929320
r_1	0.995976	1.004453
a_2	6.243932	6.116298
b_2	5.004258	4.891445
r_2	1.004928	1.000088
a_3	4.829022	4.817583
b_3	2.147494	2.340765
r_3	1.004839	1.010843
s	0.315680	-0.010000

To see this, consider the constraints in the optimisation problem (33) relating to the parameter constraints:

$$s - (|d_{i,j} - D_{i,j}| - \tau_{i,j}) \geq 0, \quad i = 1, \ldots, j, \quad j = 1, \ldots, 3.$$

Putting $s = -0.01$ and $\tau_{i,j} = 0.01$, we obtain

$$-0.01 - (|d_{i,j} - D_{i,j}| - 0.01) \geq 0, \quad i = 1, \ldots, j, \quad j = 1, \ldots, 3,$$

and thus

$$|d_{i,j} - D_{i,j}| \leq 0, \quad i = 1, \ldots, j, \quad j = 1, \ldots, 3,$$

from which (34) follows.

8 SUMMARY

In this report we have been concerned with the mathematical techniques underlying computer-aided inspection methods used for checking whether a manufactured part meets its design specification. In particular, we have described how general purpose optimisation software may be used to solve various classes of geometric tolerance assessment problem. This has been achieved by presenting in the form of generic submodules those software routines that are necessary to interface the optimisation software with the tolerance assessment problem.

The classes of problem we have considered include form assessment, profile and template matching, as well as part mating problems. A common property of these problems is that they may all be formulated as constrained optimisation problems. We have illustrated how we may include tolerances on the parameters that define the nominal form, *parameter constraints*, as well as *form constraints* that involve the measured data. Finally, we have shown that, conceptually, it is no more difficult to solve these problems where the nominal form is defined parametrically, or as a multi-component artefact, than to deal with the fundamental geometric elements such as the circle, sphere and cylinder.

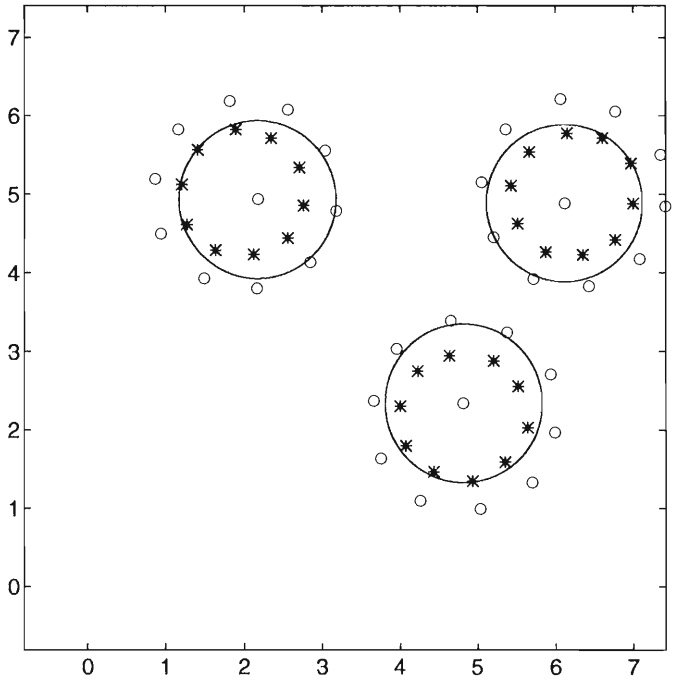


Figure 7 Solution to the separating circles problem with centre constraints for the data shown in Figure 5: the separating circles are indicated for the hole data and the transformed plug data.

9 ACKNOWLEDGEMENTS

The work reported on here was carried out as part of the project Mathematical Algorithms for Metrology 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, Dr M G Cox and Dr S A Hannaby have contributed to this work. In particular, Maurice Cox made many helpful comments on drafts of this report. Our approach for distance functions described in section 4 was influenced by helpful discussions with Prof M J D Powell, University of Cambridge.

10 REFERENCES

[1] G.T. Anthony, H.M. Anthony, B. Bittner, B.P. Butler, M.G. Cox, R. Drieschner, R. Elligsen, A.B. Forbes, H. Groß, S.A. Hannaby, P.M. Harris, and J. Kok. Chebyshev reference software for the evaluation of coordinate measuring machine data. Commission of the European Communities, BCR Report EUR 15304 EN, Luxembourg, 1993.

[2] G.T. Anthony, H.M. Anthony, B. Bittner, B.P. Butler, M.G. Cox, R. Drieschner, R. Elligsen, A.B. Forbes, H. Groß, S.A. Hannaby, P.M. Harris, and J. Kok. Chebyshev best-fit geometric elements. Technical Report DITC 221/93, National Physical Laboratory, Teddington, 1993.

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

[4] G.T. Anthony and M.G. Cox. Reliable algorithms for roundness assessment according to BS3730. In M.G. Cox and G.N. Peggs, editors, *Software for Co-ordinate Measuring Machines*, pages 30–37, National Physical Laboratory, Teddington, 1986.

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

[6] R.K. Brayton, S.W. Director, G.D. Hatchel, and L. Vidigal. A new algorithm for statistical circuit design based on quasi-Newton methods and function splitting. *IEEE Trans. Circuits and Systems*, CAS-26:784–794, 1979.

[7] British Standards Institution, London. *BS 3730: Assessment of Departure from Roundness*, 1982.

[8] B.P. Butler, 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.* To appear.

[9] D.G. Chetwynd. Linearized exchange algorithms in metrology. In M.G. Cox and G.N. Peggs, editors, *Software for Co-ordinate Measuring Machines*, pages 24–29, National Physical Laboratory, Teddington, 1986.

[10] A.R. Conn and Y. Li. An efficient algorithm for nonlinear minimax problems. Technical Report CS-88-41, University of Waterloo Computer Science Department, Waterloo, Ontario, Canada, 1989.

[11] M.G. Cox. Linear algebra support modules for approximation and other software. In J.C. Mason and M.G. Cox, editors, *Scientific Software Systems*, pages 21–29, Chapman & Hall, London, 1990.

[12] A.V. Fiacco and G.P. McCormick. *Nonlinear Programming – Sequential Unconstrained Minimization Techniques.* Society for Industrial and Applied Mathematics, Philadelphia, 1990.

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

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

[15] A.B. Forbes. Fitting an ellipse to data. Technical Report DITC 95/87, National Physical Laboratory, Teddington, 1987.

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

[17] A.B. Forbes. Least-squares best fit geometric elements. In J.C. Mason and M.G. Cox, editors, *Algorithms for Approximation II*, Chapman & Hall, London, 1990.

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

[19] A.B Forbes and P.M. Harris. Chebyshev best-fit geometric elements by mathematical programming. Technical report, National Physical Laboratory, Teddington. To appear.

[20] P.E. Gill and W. Murray, editors. *Numerical methods for constrained optimization*, Academic Press, London, 1974.

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

[22] A. Grace. *Optimization Toolbox User's Guide.* The MathWorks, Inc., Natick, Mass., 1992.

[23] *Matlab Reference Guide.* The MathWorks, Inc., Natick, Mass., 1992.

[24] C. Moler, J. Little, and S. Bangert. *Pro-Matlab for VAX/VMS Computers.* The MathWorks, Inc., Natick, Mass., 1988.

[25] W. Murray and M.L Overton. A projected Lagrangian algorithm for nonlinear minimax optimization. *SIAM Journal for Scientific and Statistical Computing*, 1(3):345–370, 1980.

[26] G.L. Nemhhauser, A.H.G. Rinnooy Kan, and M.J. Todd, editors. *Handbooks in Operations Research and Management Science, Volume 1: Optimization*, North-Holland, Amsterdam, 1989.

[27] The Numerical Algorithms Group Limited, Oxford. *The NAG Fortran Library, Mark 15*, 1991.

[28] M.R. Osborne and G.A. Watson. An algorithm for minimax approximation in the nonlinear case. *Computer Journal*, 12:63–68, 1969.

[29] M.J.D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. Technical Report DAMTP 1992/NA5, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England, 1992.

[30] G.A. Watson. The minimax solution of an overdetermined system of non-linear equations. *Journal of the Institute of Mathematics and its Applications*, 23:167–180, 1979.

[31] G.A. Watson. *Approximation Theory and Numerical Methods.* John Wiley & Sons, Chichester, 1980.