Least-squares best-fit geometric elements

Technical Report · January 1989

DOI: 10.1007/978-1-4899-3442-0_28

Alistair Barrie Forbes
National Physical Laboratory

NPL

National Physical Laboratory

NPL Report DITC 140/89
Revised edition February 1991

LEAST-SQUARES BEST-FIT GEOMETRIC ELEMENTS

Alistair B Forbes

ISSN 0262-5369

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

Approved on behalf of the Director of NPL by Dr T.G. Blaney,
Superintendent, Division of Information Technology and Computing

NPL Report DITC 140/89
Revised edition February 1991

NATIONAL PHYSICAL LABORATORY

LEAST-SQUARES BEST-FIT GEOMETRIC ELEMENTS

by

Alistair B Forbes

Division of Information Technology and Computing

Abstract

We describe algorithms for computing least-squares best-fit geometric elements to data. The elements considered are lines, planes, circles, spheres, cylinders and cones. The algorithms employ stable parametrizations of the elements and are based on sound mathematical and numerical principles. They should be of value to the manufacturers and users of coordinate measurement systems.

Keywords: algorithms, coordinate metrology, Gauss-Newton, numerical stability, parametrization.

Preface to revised edition

This revised edition contains only minor amendments to the original report published in April 1989. The reference section has been updated and the third line of formula (10.4) of page 21 corrected. Additional Notes have been included in sections 3.4 and 4.4.

Alistair B. Forbes
February 1991

Contents

1 Introduction

1.1 Aim of the report

The aim of this report is to describe algorithms for finding least-squares best-fit geometric elements to data. Such data fitting problems arise in coordinate metrology where it is required to assess a manufactured workpiece using data gathered by a coordinate measurement system (see, for example, Cox (1985)). We have tried to be concise and reasonably self-contained, with the emphasis on a straightforward presentation of the computational steps, rather than the mathematical derivation and numerical analysis of the algorithms. It is hoped that equipped with this report and access to a small number of well-known linear algebra modules, the reader will be able to produce practical and reliable best-fit routines with minimal effort.

The algorithms described in this report closely follow the guidelines drawn up in BS7172, the British Standard Guide to the Assessment of Geometric Features: Position, Size and Departure from Nominal Form, (BSI (1989)). Where possible we use the same notation as BS7172 and the definitions set out in the Guide may be of use here.

As in BS7172, the specific geometries considered in this report are:

Lines in a specified plane

Lines in 3 dimensions

Planes

Circles in a specified plane

Circles in 3 dimensions

Spheres

Cylinders

Cones

We are concerned only with establishing the least-squares best-fit reference element to cartesian data points (x, y, z). We assume that the data points are reasonably accurate and are representative of the geometric element concerned. Algorithms for partial geometries are the subject of research at NPL and elsewhere (Forbes (1989), Anthony, Anthony, Cox and Forbes (1991)).

1.2 Least-squares best-fit element

In this section we explain what we mean by 'establishing a least-squares best-fit reference element to data', first of all by considering an example of fitting a plane to data points. We assume that we have a set of m data points $(x_1, y_1, z_1), \ldots, (x_i, y_i, z_i), \ldots, (x_m, y_m, z_m)$ representing measurements from a nominally flat surface. The aim is to find the position and orientation of the plane which in some sense is closest to the data points. To achieve this aim, we need i) some way of describing the position and orientation of planes and ii) some measure of how well a plane fits the data.

The first of these is the problem of *parametrization.* We know that a plane can be uniquely specified by a point (x_0, y_0, z_0) in the plane and by the direction cosines (a, b, c) of the normal ((a, b, c) points perpendicularly to the plane and is such that $a^2 + b^2 + c^2 = 1$). Thus, any plane can be defined by six such parameters (which are not all independent of each other).

Now suppose we have a plane, specified by (x_0, y_0, z_0) and (a, b, c) as above, and we wish to measure how well the plane fits a set of data points. For a single point, (x, y, z), it is natural to take the distance d, given by

$$d = a(x - x_0) + b(y - y_0) + c(z - z_0), \tag{1.1}$$

of this point from the plane as the error in the fit. If we have many points then it is equally natural to use some aggregate of the distances from the points to the plane as a measure of the error. We note that in (1.1) d is either positive or negative according to which side of the plane the point lies and for this reason a straightforward sum of the distances cannot usually be used as a measure of closeness. Instead we take the sum of the squares of the distances as our measure. Thus, for each data point (x_i, y_i, z_i), we calculate the distance

$$d_i = a(x_i - x_0) + b(y_i - y_0) + c(z_i - z_0) \ ,$$

from the point to the plane and we form the sum

$$E = \sum_{i=1}^{n} d_i^2 \ . \tag{1.2}$$

This sum depends on the parameters (x_0, y_0, z_0) and (a, b, c) and finding the best-fit plane amounts to finding the values of the parameters which makes the sum of squares E take on its minimum value - hence the term *least-squares* fit.

In general, modelling the problem of finding the least-squares best-fit element to data follows the stages we have outlined above for the case of a plane: i) choose parameters to describe the position, orientation, size, and shape of the geometric element and ii) derive a formula for the distance of a point to the element in terms of these parameters. Given a set of data points, we can then express the sum of the squares, E, of the distances of the points to the element in terms of the parameters. To develop an element fitting algorithm, we have to find a numerical method of determining which values of the parameters make E minimal. In the next section we discuss some such methods.

1.3 Notes

1. Parametrization in covered in BS7172, section 2, while formulae for distances to geometric elements are given in appendix A of the Guide. What is meant by establishing a reference is discussed in section 5.

2. We have assumed for our least-squares fitting model that the errors in the x, y and z coordinates of the data points have equal, uncorrelated normal distributions. Element fitting to data with more general error characteristics is considered in Cox and Jones (1989) and Forbes (1990 and 1991).

2 Optimization

We suppose we have some function

$$E(\mathbf{u}) = \sum_{i=1}^{m} d_i^2(\mathbf{u}) \tag{2.1}$$

which we wish to minimise with respect to the parameters $\mathbf{u} = (u_1, \ldots, u_n)^T$. In our situation d_i represents the distance from the ith data point to the geometric element parametrized by $\mathbf{u}$. We assume that $m \geq n$.

2.1 Linear least-squares

The first case to consider is when each function d_i is a *linear* function of the parameters, so that there exist constants a_{ij} and b_i such that

$$d_i = a_{i1}u_1 + \ldots + a_{ij}u_j + \ldots + a_{in}u_n - b_i. \tag{2.2}$$

Ideally, we would want to have each $d_i = 0$, for then E will clearly take its minimum value. This requirement can be expressed as a system of m linear equations in the n unknowns $\mathbf{u}$. If we let A be the matrix whose (i,j)th element is a_{ij} and $\mathbf{b}$ the column vector whose ith element is b_i, we can write this system as

$$A\mathbf{u} = \mathbf{b}. \tag{2.3}$$

In general we will have $m > n$, so we will not be able to satisfy all the equations simultaneously; the least-squares solution is the one which minimises E. Fortunately, there exist reliable and efficient algorithms to solve (2.3) in the least-squares sense and software implementations of these algorithms are widely available. In fact, we can show that the $\mathbf{u}$ that minimises E is the solution of the square linear system of equations

$$A^TA\mathbf{u} = A^T\mathbf{b}, \tag{2.4}$$

(the so-called *normal equations*). However, the favoured algorithms used to determine the least-squares solution of (2.3) do not solve (2.4) directly but use a different approach that is more accurate in practice; see Note 2.4.

2.2 Gauss-Newton algorithm

We now consider the case where the functions d_i are *nonlinear* functions of the parameters. This is the situation for all of the element fitting problems considered here although the linear least-squares model has applications to circle and sphere fitting in particular. For the nonlinear problem we can mathematically derive equations for $\mathbf{u}$ similar to (2.4) but which are correspondingly nonlinear, and to solve such a system we require an iterative type of algorithm. To motivate how such an algorithm works, we first describe the Newton algorithm for finding the value u^* for which a nonlinear function $f(u)$ of one parameter u is equal to zero. We should say that mathematically this is a non-trivial problem even for simple functions like polynomials.

The situation is described in Figure 2.1. We suppose that we have some estimate u_0 of where the graph of $f(u)$ crosses the u-axis. Then, i) we evaluate $f(u_0)$ and $f'(u_0)$; ii) we

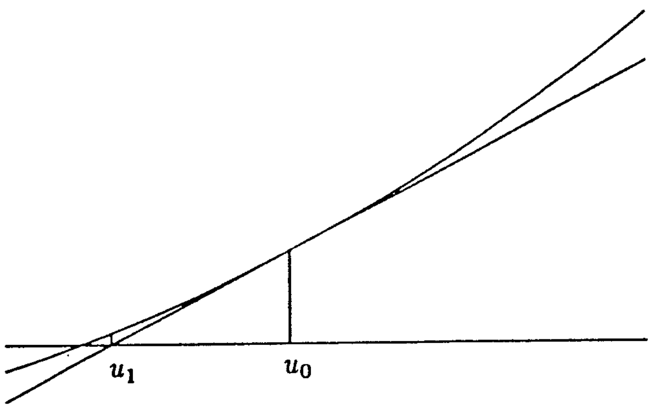

Figure 2.1: One iteration of the Newton algorithm for computing a zero of a function.

form the tangent line to the graph at the point $(u_0, f(u_0))$: we know from calculus that this line has slope $f'(u_0)$ and passes through the point $(u_0, f(u_0))$; and iii) we find the point u_1 where the tangent line crosses the u-axis:

$$u_1 = u_0 + p, \quad \text{where} \quad p = -f(u_0)/f'(u_0)\ , \tag{2.5}$$

and use this as our new estimate of where f crosses the axis. We can now repeat the process (assuming it is successful) until we are satisfied we are close enough to u^*.

The main ingredients of the algorithm in general terms are a) given an approximate solution, we linearise the problem, perhaps using calculus (this corresponds to replacing the function by its tangent); b) we solve the linear version of the problem (find where the tangent crosses the axis); and c) we use the solution of the linear problem to update our estimate of the solution.

The Gauss-Newton algorithm for finding the minimum of a sum of squares E follows the scheme outlined above. Suppose we have some estimate $\mathbf{u}$ of the solution $\mathbf{u}^*$. We solve a *linear* least-squares system of the form

$$J\mathbf{p} = -\mathbf{d}\ , \tag{2.6}$$

where: J is the $m \times n$ *Jacobian* matrix whose ith row is the gradient of d_i with respect to the parameters $\mathbf{u}$, i.e.,

$$J_{ij} = \frac{\partial d_i}{\partial u_j} \tag{2.7}$$

evaluated at $\mathbf{u}$; and the ith component of $\mathbf{d}$ is $d_i(\mathbf{u})$. Finally, we update our estimate of the solution parameters according to

$$\mathbf{u} := \mathbf{u} + \mathbf{p}\ . \tag{2.8}$$

These steps are repeated until the estimates have been judged to have converged to the solution $\mathbf{u}^*$. Three criteria are relevant for testing for convergence:

1) the change in the sum of the squares E from iteration to iteration should be small;

2) the size of the update, measured for instance by $(\mathbf{p}^T\mathbf{p})^{1/2}$, should be small; and

3) the partial derivatives of E with respect to the optimization parameters, measured for instance by $(\mathbf{g}^T\mathbf{g})^{1/2}$ where $\mathbf{g} = J^T\mathbf{d}$, should be small.

Various things can go wrong with the algorithm. If the estimate $\mathbf{u}$ is poor the subsequent estimate may be worse and so on - this behaviour is called divergence. If the data is very inaccurate, then the algorithm may take many iterations (and computation time) to converge. If the data is unrepresentative, for example data lying nominally on only a small arc of the circle, convergence can again be slow and, unless we have good starting values, the algorithm is prone to divergence. In an extreme case, the Jacobian matrix J will become rank deficient and the system (2.6) will not have a well-defined solution. However, it is our experience that given good starting values and reasonably accurate data which is representative of the element to be fitted, convergence to the solution is rapid.

2.3 Eigenvectors and Singular Value Decomposition

Our element fitting algorithms need one further concept. Given a square matrix B, an *eigenvector* $\mathbf{u}$ of B is such that

$$B\mathbf{u} = \lambda\mathbf{u}, \tag{2.9}$$

for some *eigenvalue* λ. There are well established algorithms for finding the eigenvectors of matrices and software implementations of these algorithms are widely available. We are often interested in the case where

$$B = A^T A, \tag{2.10}$$

(cf. (2.4)) for some $m \times n$ rectangular matrix A (where $m \geq n$). In this situation, we can arrive numerically more stably at the eigenvectors of A from a singular value decomposition (SVD) of the matrix A. Without going into too much detail, A can be written as a product

$$A = USV^T, \tag{2.11}$$

with U and V orthogonal matrices and S a diagonal matrix containing the singular values of A. If B is as in (2.10), the squares of the diagonal elements of S are the eigenvalues of B and the columns of V are the corresponding eigenvectors. Again software implementations of algorithms to find the SVD of a matrix are widely available. The software generally calculates the eigenvalues or singular values along with the associated vectors.

2.4 Notes

1. Details of algorithms for linear least-squares and SVD problems can be found for example in Golub and Van Loan (1983, Chapter 6). Algorithms are available for example in the NAG, LINPACK and DASL libraries (Ford, Bentley, Du Croz and Hague (1979), Dongarra, Moler, Bunch and Stewart (1979), Dongarra and Grosse (1987), Anthony and Cox (1987)). The Gauss-Newton algorithm is described in more detail in Gill, Murray and Wright (1981, section 4.7) and Fletcher (1980, Chapter 6). Optimization algorithms to solve general nonlinear least-squares problems can be found for example in the NAG and NOSL libraries (Hodson, Jones and Long (1983)).

2. The most stable algorithms to form the linear least-squares solution of (2.3) are based on a factorisation of the matrix A. These algorithms construct an orthogonal matrix Q, $Q^TQ = I$ such that $Q^TA = R$ is an upper-triangular matrix. $\mathbf{u}$ is then found from the solution of $R\mathbf{u} = Q^T\mathbf{b}$. In many implementations of these algorithms the matrix Q need not be stored; using a Givens plane rotation strategy (Cox (1981)), the matrix A can be processed one row at a time and only the upper-triangular matrix R has to be stored.

3. Various strategies can be used to improve the performance of the Gauss-Newton algorithm for less favourable optimization problems. One method regards the vector $\mathbf{p}$ as a search direction and a one variable optimization algorithm is applied to the function $E(\mathbf{u}+t\mathbf{p})$ to find a value of t which gives an acceptable decrease in E. The trust region method monitors how well the actual decrease in E compares with the expected decrease, and if necessary will adjust the update step to produce an improved reduction.

Element fitting algorithms

In the subsequent sections we present algorithms for finding the parameters of the eight geometric elements listed in section 1.1. For each case we describe how to parametrize the element, give the appropriate formula for the distance of a point to the element and then present an algorithm for computing the best-fit element to data. We have tried to keep the sections independent of each other at the expense of some duplication of material.

Each algorithm requires a minimum number of data points as given in Table 2.1. For the best results, the algorithms require that the data is fully representative of the corresponding geometric element, i.e. an adequate (at least twice the minimum) number of data points well distributed over the surface of the element. In general, the more data points there are, the more reliable will be the results of the assessment.

To maintain numerical accuracy, it is important that the mean position (centroid) of the data points, $(\bar{x}, \bar{y}, \bar{z})$, is close to the origin $(0, 0, 0)$. The easiest way to achieve this is to calculate $(\bar{x}, \bar{y}, \bar{z})$ and then translate (a copy of) the data so that the new centroid lies at the origin:

$$\begin{aligned} \bar{x} &= \frac{1}{m}\sum_{i=1}^{m} x_i, \quad \text{etc.}, \\ (x_i, y_i, z_i) &:= (x_i, y_i, z_i) - (\bar{x}, \bar{y}, \bar{z}), \quad i = 1, \ldots, m. \end{aligned}$$

The line and plane fitting problems reduce to finding the eigenvectors of matrices, while all the other fitting problems can be solved using the Gauss-Newton algorithm. We also describe circle and sphere fitting models that can be presented as linear least-squares problems. The algorithm for finding the best-fit circle in 3 dimensions is a little complicated and we have postponed it until after the cylinder case.

element	min. no.
line	2
plane	3
circle	3
sphere	4
cylinder	5
cone	6

Table 2.1: The minimum number of data points required by each element fitting algorithm.

3 Lines in a specified plane

We assume that we wish to fit a line to m points (x_i, y_i) in the xy-plane, where $m \geq 2$; the case of a line in an arbitrary plane can be reduced to this case by rotating and translating the data points; how this can be achieved is discussed in section 8.

3.1 Parametrization

We specify a line in the plane by

i) a point (x_0, y_0) on the line and

ii) the direction cosines (a, b) where $a^2 + b^2 = 1$.

We note that any point (x, y) on the line satisfies

$$(x, y) = (x_0, y_0) + t(a, b)$$

for some value of t.

3.2 Distance from a point to a line in the plane

Given a line specified by x_0, y_0, a and b as above, the signed[1] distance from point (x_i, y_i) to the line is found from

$$d_i = b(x_i - x_0) - a(y_i - y_0)\,. \tag{3.1}$$

3.3 Algorithm description

The best-fit line, L, passes through the centroid $(\bar{x}, \bar{y})$ of the data and this determines a point on L; we also have to find the direction cosines of L. For this we can show that (a, b) is the eigenvector associated with the largest eigenvalue of

$$B = A^T A\,,$$

where A is the $m \times 2$ matrix whose ith row is $(x_i - \bar{x}, y_i - \bar{y})$. Alternatively, (a, b) is the singular vector associated with the largest singular value of A. Thus, an algorithm to find the best-fit line to data in the plane follows the steps below:

I calculate the centroid $(\bar{x}, \bar{y})$;

II form the matrix A;

III find the SVD of A and choose the singular vector (a, b) corresponding to the largest singular value.

The best-fit line is specified by $\bar{x}$, $\bar{y}$, a and b.

[1]See section 3.4.

3.4 Notes

1. If we write (a, b) as $(\cos\theta, \sin\theta)$ for some angle θ, then the optimal a and b can alternatively be found from

$$\tan 2\theta = \frac{2\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(y_i - \bar{y})^2 - (x_i - \bar{x})^2}.$$

However this method can suffer from loss of numerical accuracy and is not recommended.

2. If we replace the direction cosines (a, b) by $(-a, -b)$ we specify the same line but change the sign of d_i given by (3.1); similar remarks apply to fitting planes. The algorithms here do not assume or depend on a particular sign convention; see Anthony *et al.* (1991) for a further discussion on the parametrization of lines and planes.

4 Lines in 3 dimensions

We assume that we wish to fit a line to m points (x_i, y_i, z_i), where $m \geq 2$.

4.1 Parametrization

We specify a line in the plane by

i) a point (x_0, y_0, z_0) on the line and

ii) the direction cosines (a, b, c) where $a^2 + b^2 + c^2 = 1$.

We note that any point (x, y, z) on the line satisfies

$$(x, y, z) = (x_0, y_0, z_0) + t(a, b, c) \tag{4.1}$$

for some value of t.

4.2 Distance from a point to a line in 3 dimensions

Given a line specified by x_0, y_0, z_0, a, b and c as above, the distance from point (x_i, y_i, z_i) to the line is found from

$$d_i = \sqrt{[u_i^2 + v_i^2 + w_i^2]}, \tag{4.2}$$

where

$$\begin{aligned} u_i &= c(y_i - y_0) - b(z_i - z_0), \\ v_i &= a(z_i - z_0) - c(x_i - x_0), \\ w_i &= b(x_i - x_0) - a(y_i - y_0). \end{aligned}$$

The reader may note that (u_i, v_i, w_i) is the vector cross-product of $[(x_i, y_i, z_i) - (x_0, y_0, z_0)]$ with (a, b, c).

4.3 Algorithm description

The best-fit line, L, passes through the centroid $(\bar{x}, \bar{y}, \bar{z})$ of the data and this specifies a point on L; we also have to find the direction cosines of L. For this we can show that (a, b, c) is the eigenvector associated with the largest eigenvalue of

$$B = A^T A,$$

where A is the $m \times 3$ matrix whose ith row is $(x_i - \bar{x}, y_i - \bar{y}, z_i - \bar{z})$. Alternatively, (a, b, c) is the singular vector associated with the largest singular value of A. Thus, an algorithm to find the best-fit line to data in 3 dimensions follows the steps below:

I calculate the centroid $(\bar{x}, \bar{y}, \bar{z})$;

II form the matrix A;

III find the SVD of A and choose the singular vector (a, b, c) corresponding to the largest singular value.

The best-fit line is specified by $\bar{x}$, $\bar{y}$, $\bar{z}$, a, b and c.

4.4 Notes

1. As in the case of lines in the plane, the direction cosines are determined only up to sign. However in this case there is no comparable notion of a *signed* distance from a point to a line in 3 dimensions and d_i formed as in (4.2) is always non-negative. The parametrization of lines in 3 dimensions is discussed further in section 8, and also in Anthony *et al.* (1991).

5 Planes

We assume that we wish to fit a plane to m points (x_i, y_i, z_i), where $m \geq 3$.

5.1 Parametrization

We specify a plane by

i) a point (x_0, y_0, z_0) on the plane and

ii) the direction cosines (a, b, c) of the normal to the plane, where $a^2 + b^2 + c^2 = 1$.

We note that any point (x, y, z) on the plane satisfies

$$a(x - x_0) + b(y - y_0) + c(z - z_0) = 0.$$

5.2 Distance from a point to a plane

Given a plane specified by x_0, y_0, z_0, a, b and c as above, the signed[2] distance from point (x_i, y_i, z_i) to the plane is found from

$$d_i = a(x_i - x_0) + b(y_i - y_0) + c(z_i - z_0). \tag{5.1}$$

5.3 Algorithm description

The best-fit plane, P, passes through the centroid $(\bar{x}, \bar{y}, \bar{z})$ of the data and this specifies a point on P; we also have to find the direction cosines of P. For this we can show that (a, b, c) is the eigenvector associated with the smallest eigenvalue of

$$B = A^T A,$$

where A is the $m \times 3$ matrix whose ith row is $(x_i - \bar{x}, y_i - \bar{y}, z_i - \bar{z})$. Alternatively, (a, b, c) is the singular vector associated with the smallest singular value of A. Thus, an algorithm to find the best-fit line to data in 3 dimensions follows the steps below:

I calculate the centroid $(\bar{x}, \bar{y}, \bar{z})$;

II form the matrix A;

III find the SVD of A and choose the singular vector (a, b, c) corresponding to the smallest singular value.

The best-fit plane is specified by $\bar{x}$, $\bar{y}$, $\bar{z}$, a, b and c.

5.4 Notes

1. There is a close relationship between the best-fit line and plane to points in 3 dimensions stemming from the fact that, given a line specified by (x_0, y_0, z_0) and (a, b, c) and a point (x_i, y_i, z_i), we have

$$(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2 = e_i^2 + f_i^2,$$

where e_i is the distance from (x_i, y_i, z_i) to the line and f_i is the distance from (x_i, y_i, z_i) to the plane specified by (x_0, y_0, z_0) and (a, b, c). This equation can be used to show why the best-fit plane and line correspond respectively to the eigenvectors associated with the smallest and largest eigenvalues of A.

2. The formulation of the plane and line fitting problems as eigenvalue problems is necessitated by the requirement on the direction cosines that $a^2 + b^2 + c^2 = 1$. For cylinder and cone fitting it proves to be advantageous to impose a different constraint on (a, b, c); see section 8.

[2]See section 3.4.

6 Circles in a specified plane

We assume that we wish to fit a circle to m points (x_i, y_i) in the xy-plane, where $m \geq 3$; the case of the circle in an arbitrary plane can be reduced to this case by rotating and translating the data: see sections 8 and 10.4.

6.1 Parametrization

We specify a circle in the plane by

i) its centre (x_0, y_0) and

ii) its radius r.

We note that any point (x, y) on the circle satisfies

$$(x - x_0)^2 + (y - y_0)^2 = r^2 .$$

6.2 Distance from a point to a circle in the plane

Given a circle specified by x_0, y_0, and r as above, the distance from a point (x_i, y_i) to the circle is found from

$$d_i = r_i - r, \tag{6.1}$$

where

$$r_i = \sqrt{[(x_i - x_0)^2 + (y_i - y_0)^2]}.$$

The elements of the Jacobian matrix J are found from the partial derivatives of d_i with respect to the parameters x_0, y_0 and r given by

$$\begin{aligned} \frac{\partial d_i}{\partial x_0} &= -(x_i - x_0)/r_i, \\ \frac{\partial d_i}{\partial y_0} &= -(y_i - y_0)/r_i, \\ \frac{\partial d_i}{\partial r} &= -1. \end{aligned} \tag{6.2}$$

6.3 Algorithm description

The algorithm to find the best-fit circle is based on the Gauss-Newton algorithm described in section 2.2. We assume that we have estimates (x_0, y_0) and r for the circle centre and radius. In section 6.4, we show how we can find good estimates of these parameters. One iteration of the Gauss-Newton algorithm follows the steps below:

I form the right hand side vector $\mathbf{d}$ and Jacobian matrix J according to (6.1) and (6.2);

II solve the linear least-squares system

$$J \begin{bmatrix} p_{x_0} \\ p_{y_0} \\ p_r \end{bmatrix} = -\mathbf{d};$$

III update the parameter estimates according to

$$\begin{aligned} x_0 &:= x_0 + p_{x_0}, \\ y_0 &:= y_0 + p_{y_0}, \\ r &:= r + p_r. \end{aligned}$$

These steps are repeated until the algorithm has converged.

6.4 Linear least-squares circle

In this section we outline a circle fitting model that can be solved as a linear least-squares problem. For accurate data, this model should give a best-fit circle which agrees very closely with the best-fit circle found according to the full nonlinear model described above; for less accurate data, the approximate model can be used to furnish good initial estimates of the centre and radius parameters.

In the approximate model we minimise

$$F = \sum_{i=1}^{m} f_i^2,$$

where

$$f_i = r_i^2 - r^2.$$

By a change of parameters, we can make f into a linear function

$$\begin{aligned} f_i &= (x_i - x_0)^2 + (y_i - y_0)^2 - r^2, \\ &= -2x_i x_0 - 2y_i y_0 + (x_0^2 + y_0^2 - r^2) + (x_i^2 + y_i^2), \end{aligned} \qquad (6.3)$$

of x_0, y_0 and $\rho = x_0^2 + y_0^2 - r^2$. To minimise F, therefore, we solve the linear least-squares system

$$A \begin{bmatrix} x_0 \\ y_0 \\ \rho \end{bmatrix} = \mathbf{b},$$

where the elements of the ith row of A are the coefficients $(2x_i, 2y_i, -1)$ and the ith element of $\mathbf{b}$ is $x_i^2 + y_i^2$. We recover an estimate of r from

$$r = \sqrt{[x_0^2 + y_0^2 - \rho]}.$$

6.5 Notes

1. Circle fitting is discussed in Anthony and Cox (1986), wherein algorithms to find the least-squares, linear least-squares, maximum inscribed, minimum circumscribed and minimum zone best-fit circles are described. Finding best-fit circles to partial arcs is considered in Forbes (1989).

7 Spheres

We assume that we wish to fit a sphere to m points (x_i, y_i, z_i), where $m \geq 4$. We should point out that this section is just the three dimensional version of section 6.

7.1 Parametrization

We specify a sphere by

i) its centre (x_0, y_0, z_0) and

ii) its radius r.

We note that any point (x, y, z) on the sphere satisfies

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2 .$$

7.2 Distance from a point to a sphere

Given a sphere specified by x_0, y_0, z_0 and r as above, the distance from a point (x_i, y_i, z_i) to the sphere is found from

$$d_i = r_i - r, \tag{7.1}$$

where

$$r_i = \sqrt{[(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2]}.$$

The elements of the Jacobian matrix J are found from the partial derivatives of d_i with respect to the parameters x_0, y_0, z_0 and r given by

$$\begin{aligned} \frac{\partial d_i}{\partial x_0} &= -(x_i - x_0)/r_i, \\ \frac{\partial d_i}{\partial y_0} &= -(y_i - y_0)/r_i, \\ \frac{\partial d_i}{\partial z_0} &= -(z_i - z_0)/r_i, \\ \frac{\partial d_i}{\partial r} &= -1. \end{aligned} \tag{7.2}$$

7.3 Algorithm description

The algorithm to find the best-fit sphere is based on the Gauss-Newton algorithm described in section 2.2. We assume that we have estimates (x_0, y_0, z_0) and r for the sphere centre and radius. In section 7.4, we show how we can find good estimates of these parameters. One iteration of the Gauss-Newton algorithm follows the steps below:

I form the right hand side vector $\mathbf{d}$ and Jacobian matrix J according to (7.1) and (7.2);

II solve the linear least-squares system

$$J \begin{bmatrix} p_{x_0} \\ p_{y_0} \\ p_{z_0} \\ p_r \end{bmatrix} = -\mathbf{d};$$

III update the parameter estimates according to

$$\begin{aligned} x_0 &:= x_0 + p_{x_0}, \\ y_0 &:= y_0 + p_{y_0}, \\ z_0 &:= z_0 + p_{z_0}, \\ r &:= r + p_r. \end{aligned}$$

These steps are repeated until the algorithm has converged.

7.4 Linear least-squares sphere

In this section we outline a sphere fitting model that can be solved as a linear least-squares problem. For accurate data, this model should give a best-fit sphere which agrees very closely with the best-fit sphere found according to the full nonlinear model described above; for less accurate data, the approximate model can be used to furnish good initial estimates of the centre and radius parameters.

In the approximate model we minimise

$$F = \sum_{i=1}^{m} f_i^2$$

where

$$f_i = r_i^2 - r^2.$$

By a change of parameters, we can make f into a linear function

$$\begin{aligned} f_i &= (x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2 - r^2, \\ &= -2x_i x_0 - 2y_i y_0 - 2z_i z_0 + (x_0^2 + y_0^2 + z_0^2 - r^2) + (x_i^2 + y_i^2 + z_i^2), \end{aligned} \tag{7.3}$$

of x_0, y_0, z_0 and $\rho = x_0^2 + y_0^2 + z_0^2 - r^2$. To minimise F, therefore, we solve the linear least-squares system

$$A \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ \rho \end{bmatrix} = \mathbf{b},$$

where the elements of the ith row of A are the coefficients $(2x_i, 2y_i, 2z_i, -1)$ and the ith element of $\mathbf{b}$ is $x_i^2 + y_i^2 + z_i^2$. We recover an estimate of r from

$$r = \sqrt{[x_0^2 + y_0^2 + z_0^2 - \rho]}.$$

7.5 Notes

1. Sphere fitting is discussed in Boffey, Cox, Delves and Pursglove (1990), wherein algorithms to find the least-squares, maximum inscribed, minimum circumscribed and minimum zone spheres are described. In some applications, for example in lens making, it is required to fit a sphere to points representing only a small patch of the sphere surface; this case is considered in Forbes (1989).

8 Gauss-Newton strategy for circles in 3 dimensions, cylinders and cones

The remaining three elements, circles in 3 dimensions, cylinders and cones, all require that an axis, i.e., a line in 3 dimensions, be parametrized. We have indeed discussed this case in section 4.1 but we now need to consider the problem a little more closely.

We have said that a line can be specified by giving a point on the line and the direction cosines, so that at first sight it would appear that six numbers are required to describe a line. However the constraint, $a^2 + b^2 + c^2 = 1$, on the length of the direction vector means that as soon as two of the components have been assigned, the third is determined (apart from its numerical sign); below we consider an alternative constraint on the length of the direction vector. Similarly, the fact that a point specifying the position of the line has itself to lie on the line and thus satisfy an equation of the form (4.1) means again that if two components of (x_0, y_0, z_0) are assigned the third is also determined. Thus a line in three dimensions can be specified by four parameters along with unambiguous rules (or constraints) which give the other two parameters. We consider two such rules.

Rule 1. We represent a direction by a vector of the form $(a, b, 1)$.

Rule 2. Given that the direction is given by $(a, b, 1)$, we require that

$$z_0 = -ax_0 - by_0. \tag{8.1}$$

Rule 1 is the simplest of all the rules that could be applied: we have a and b as parameters and we know that c is always 1. This rule is effective for lines that are nearly vertical, but suffers from instability for lines that are nearly horizontal - a small change in the orientation of a near horizontal line would result in a large change in a and b; moreover, the representation breaks down for lines exactly perpendicular to the z axis. For such lines we could specify that a or b should be 1 instead and use the other two components as parameters.

Rule 2 is a sensible rule to assign z_0, given x_0 and y_0, and works well as long as $(a, b, 1)$ is a suitable parametrization of direction. It is motivated by the fact that it specifies the point on the axis nearest the origin. We assume that the data centroid is at or near the origin and consequently (x_0, y_0, z_0) so constrained will have the numerically advantageous property of being near the centroid.

We can say, then, that for nearly vertical lines, these two rules give stable parametrizations in terms of the four parameters a, b, x_0, and y_0.

We now consider a second feature of fitting with these three elements. We suppose for the moment that the element axis is indeed nearly vertical so that we can parametrize it with the four parameters above. As we can see from (4.2) the expression for the distance from a point to an axis is quite complicated. To implement a Gauss-Newton algorithm, we have also to find the derivatives of this distance with respect to the parameters and, as can be imagined, this gives rise to rather unwieldy expressions which take a significant amount of computer time to evaluate. However, if the axis is exactly vertical and passes through the origin, i.e., $a = b = x_0 = y_0 = 0$, all of the expressions become vastly simplified.

These remarks suggest the following strategy for implementing a Gauss-Newton algorithm. We iterate as usual, except that at the beginning of each iteration, we translate and rotate a copy of the data so that the trial best-fit element (i.e. the element corresponding to the current estimates of the parameters) has a vertical axis passing through the origin.

This means that when we come to evaluate the Jacobian matrix, we can utilise the special orientation to simplify the calculations. At the end of an iteration we use the inverse rotation and translation transformations to update the parametrizing vectors (x_0, y_0, z_0) and (a, b, c) and thus determine the new position and orientation of the axis. A further advantage of this scheme is that we are always using the parametrization given by Rules 1 and 2 in ideal circumstances.

We can rotate a point (x, y, z) by applying a 3×3 rotation matrix U to the vector $(x, y, z)^T$; the inverse rotation can be performed using the transpose of U. Thus:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = U \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad (8.2)$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = U^T \begin{bmatrix} u \\ v \\ w \end{bmatrix}. \qquad (8.3)$$

There are various ways we can construct a rotation matrix U to rotate a point so that it lies on the z-axis. One of the simplest is to have U of the form

$$U = \begin{bmatrix} c_2 & 0 & s_2 \\ 0 & 1 & 0 \\ -s_2 & 0 & c_2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_1 & s_1 \\ 0 & -s_1 & c_1 \end{bmatrix}, \qquad (8.4)$$

where $c_i = \cos\theta_i$ and $s_i = \sin\theta_i$, $i = 1, 2$. For example, if we want to rotate (a, b, c) to a point on the z-axis, we choose θ_1 so that $bc_1 + cs_1 = 0$ and subsequently choose θ_2 so that $ac_2 + (cc_1 - bs_1)s_2 = 0$.

9 Cylinders

We assume that we wish to fit a cylinder to m points (x_i, y_i, z_i), where $m \geq 5$.

9.1 Parametrization

We specify a cylinder by

i) a point (x_0, y_0, z_0) on its axis,

ii) a vector (a, b, c) pointing along the axis and

iii) its radius r.

To parametrize a cylinder properly we need a systematic way of deciding which point on the axis to choose, along with a constraint on (a, b, c). Following section 8, we can do this for nearly vertical cylinders by setting

$$\begin{aligned} z_0 &= -ax_0 - by_0, \\ c &= 1. \end{aligned} \tag{9.1}$$

9.2 Distance from a point to a cylinder

Given a cylinder specified by x_0, y_0, z_0, a, b, c, and r as above, the distance from a point (x_i, y_i, z_i) to the cylinder is found from

$$d_i = r_i - r, \tag{9.2}$$

where

$$r_i = \frac{\sqrt{[u_i^2 + v_i^2 + w_i^2]}}{\sqrt{[a^2 + b^2 + c^2]}}, \tag{9.3}$$

with

$$\begin{aligned} u_i &= c(y_i - y_0) - b(z_i - z_0), \\ v_i &= a(z_i - z_0) - c(x_i - x_0), \\ w_i &= b(x_i - x_0) - a(y_i - y_0). \end{aligned} \tag{9.4}$$

r_i is the distance of the ith point to the cylinder axis; the reader may note that (u_i, v_i, w_i) can be expressed as a vector cross-product of $[(x_i, y_i, z_i) - (x_0, y_0, z_0)]$ with (a, b, c).

If we employ the constraints (9.1), d_i is a function of the five parameters x_0, y_0, a, b and r, and to implement a Gauss-Newton algorithm to minimise the sum to squares of the distances we need to calculate the partial derivatives of d_i with respect to these five parameters. The expressions are algebraically complicated, but for the special case when $x_0 = y_0 = a = b = 0$, they simplify to

$$r_i = \sqrt{[x_i^2 + y_i^2]} \tag{9.5}$$

and

$$\begin{aligned}
\frac{\partial d_i}{\partial x_0} &= -x_i/r_i, \\
\frac{\partial d_i}{\partial y_0} &= -y_i/r_i, \\
\frac{\partial d_i}{\partial a} &= -x_i z_i/r_i, \\
\frac{\partial d_i}{\partial b} &= -y_i z_i/r_i, \\
\frac{\partial d_i}{\partial r} &= -1.
\end{aligned} \tag{9.6}$$

9.3 Algorithm description

The algorithm to find the best-fit cylinder is based on the Gauss-Newton strategy outlined in sections 2.2 and 8. Given estimates of (x_0, y_0, z_0), (a, b, c) and r, one iteration of the algorithm follows the steps below:

I translate (a copy of) the data so that the point on the axis lies at the origin:

$$(x_i, y_i, z_i) := (x_i, y_i, z_i) - (x_0, y_0, z_0);$$

II transform the data by a rotation matrix U which rotates (a, b, c) to a point on the z-axis:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} := U \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix};$$

III form the right hand side vector **d** and Jacobian matrix according to (9.2), (9.5) and (9.6);

IV solve the linear least-squares system

$$J \begin{bmatrix} p_{x_0} \\ p_{y_0} \\ p_a \\ p_b \\ p_r \end{bmatrix} = -\mathbf{d};$$

V update the parameter estimates according to

$$\begin{aligned}
\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} &:= \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + U^T \begin{bmatrix} p_{x_0} \\ p_{y_0} \\ -p_{x_0} p_a - p_{y_0} p_b \end{bmatrix}; \\
\begin{bmatrix} a \\ b \\ c \end{bmatrix} &:= U^T \begin{bmatrix} p_a \\ p_b \\ 1 \end{bmatrix}; \\
r &:= r + p_r.
\end{aligned}$$

These steps are repeated until the algorithm has converged. In step I, we always start with (a copy of) the original data set rather than a transformed set from a previous iteration. If we wish to have (x_0, y_0, z_0) represent the point on the line nearest the origin, then we introduce one further step:

VI

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} := \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} - \frac{ax_0 + by_0 + cz_0}{a^2 + b^2 + c^2} \begin{bmatrix} a \\ b \\ c \end{bmatrix}.$$

9.4 Notes

1. There appears to be no straightforward method for obtaining initial estimates of the parameters. If there are 9 or more data points, we can fit a general quadric,

$$Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J = 0,$$

to the data and derive estimates of (x_0, y_0, z_0), (a, b, c) and r from the solution parameters A, B, $\ldots, J$. This fitting problem can be posed as a linear least-squares or eigenvalue problem. If there are fewer than 9 points then it is not clear what can be done.

If there are estimates of (a, b, c), then we can rotate the data so that the trial axis is vertical and fit a circle to the x- and y-coordinates of the rotated points to obtain estimates of (x_0, y_0, z_0) and r. In many measurement situations such estimates of (a, b, c) are available from the approximately known orientation of the workpiece with respect to the coordinate system of the measuring machine.

10 Circles in 3 dimensions

We assume that we wish to fit a circle to m points (x_i, y_i, z_i), where $m \geq 3$, in 3 dimensions. This is in fact the most complicated of all the fitting problems considered in this report and the reader may wish to look at the cylinder case first (section 9).

10.1 Parametrization

We specify a circle in 3 dimensions by

i) its centre (x_0, y_0, z_0),

ii) its radius r and

iii) a vector (a, b, c) normal to the plane containing the circle.

To parametrize the circle properly we need to constrain the length of the vector (a, b, c). Following section 8, we can do this for circles lying in nearly horizontal planes by setting $c = 1$.

10.2 Distance from a point to a circle in 3 dimensions

Given a circle specified by x_0, y_0, z_0, r, a, b and c as above, the distance from a point (x_i, y_i, z_i) to the circle is found from

$$d_i^2 = e_i^2 + f_i^2. \tag{10.1}$$

In this equation, e_i is the distance from (x_i, y_i, z_i) to the cylinder specified by (x_0, y_0, z_0), r and $(a, b, c)/\surd[a^2 + b^2 + c^2]$ (see (9.2)) and f_i is the distance from (x_i, y_i, z_i) to the plane specified by (x_0, y_0, z_0) and (a, b, c) (see (5.1)). If the circle lies in the xy-plane, then we have

$$\begin{aligned} e_i &= \surd[(x_i - x_0)^2 + (y_i - y_0)^2] - r, \\ f_i &= z_i. \end{aligned} \tag{10.2}$$

We note that in (10.1), d_i^2 is itself a sum of two squares, and for this reason we model the element fitting problem as minimising

$$E = \sum_{i=1}^{m} d_i^2 = \sum_{i=1}^{m} e_i^2 + \sum_{i=1}^{m} f_i^2. \tag{10.3}$$

If we employ the constraint $c = 1$, d_i is a function of the six parameters x_0, y_0, z_0, a, b and r. To implement a Gauss-Newton algorithm with E as in (10.3), we need to calculate the derivatives of e_i and f_i with respect to these six optimization parameters. The expressions are lengthy, but for the special case $x_0 = y_0 = z_0 = a = b = 0$, we have

$$\begin{aligned} \frac{\partial e_i}{\partial x_0} &= -x_i/r_i, & \frac{\partial f_i}{\partial x_0} &= 0, \\ \frac{\partial e_i}{\partial y_0} &= -y_i/r_i, & \frac{\partial f_i}{\partial y_0} &= 0, \\ \frac{\partial e_i}{\partial z_0} &= 0, & \frac{\partial f_i}{\partial z_0} &= -1, \\ \frac{\partial e_i}{\partial a} &= -x_i z_i/r_i, & \frac{\partial f_i}{\partial a} &= x_i, \\ \frac{\partial e_i}{\partial b} &= -y_i z_i/r_i, & \frac{\partial f_i}{\partial b} &= y_i, \\ \frac{\partial e_i}{\partial r} &= -1, & \frac{\partial f_i}{\partial r} &= 0. \end{aligned} \tag{10.4}$$

We note that three of the f_i derivatives are zero and we can use this fact to make the solution of the Jacobian linear least-squares system more efficient.

10.3 Algorithm description

The algorithm to find the best-fit circle in 3 dimensions is based on the Gauss-Newton strategy outlined in sections 2.2 and 8. Given estimates of (x_0, y_0, z_0), (a, b, c) and r, one iteration of the algorithm follows the steps below:

I translate (a copy of) the data so that the centre estimate lies at the origin:

$$(x_i, y_i, z_i) := (x_i, y_i, z_i) - (x_0, y_0, z_0);$$

II transform the data by a rotation matrix U which rotates (a, b, c) to a point on the z-axis:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} := U \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix};$$

III form the right hand side vectors **e** and **f** according to (10.2);

IV form two Jacobian matrices J_e and J_f where: J_e is the matrix of partial derivatives of the e_i and J_f is the matrix of partial derivatives of the f_i, with the partial derivatives calculated according to (10.4);

V solve the linear least-squares system

$$\begin{bmatrix} J_e \\ J_f \end{bmatrix} \begin{bmatrix} p_{x_0} \\ p_{y_0} \\ p_{z_0} \\ p_a \\ p_b \\ p_r \end{bmatrix} = \begin{bmatrix} -\mathbf{e} \\ -\mathbf{f} \end{bmatrix};$$

VI update the parameter estimates according to

$$\begin{aligned} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} &:= \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + U^T \begin{bmatrix} p_{x_0} \\ p_{y_0} \\ p_{z_0} \end{bmatrix}; \\ \begin{bmatrix} a \\ b \\ c \end{bmatrix} &:= U^T \begin{bmatrix} p_a \\ p_b \\ 1 \end{bmatrix}; \\ r &:= r + p_r. \end{aligned}$$

These steps are repeated until the algorithm has converged. In step I we use (a copy of) the original data points rather than a transformed set from a previous iteration.

10.4 Initial estimation of the parameters

We can use plane fitting (section 5) and linear least-squares circle fitting (section 6.4) algorithms to arrive at initial estimates for (x_0, y_0, z_0), (a, b, c) and r by following the steps below:

I fit a plane to the data points to get estimates of (a, b, c) along with the centroid $(\bar{x}, \bar{y}, \bar{z})$;

II translate the data points so that $(\bar{x}, \bar{y}, \bar{z})$ is now at (0,0,0);

III rotate the data points by a rotation matrix U which transforms (a, b, c) to a point on the z-axis;

IV use the linear least-squares circle fitting algorithm to find an approximate best-fit circle specified by centre $(\hat{x}_0, \hat{y}_0)$ and radius r to the x and y coordinates of the rotated data points;

V obtain the estimate of (x_0, y_0, z_0) from

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} = \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{bmatrix} + U^T \begin{bmatrix} \hat{x}_0 \\ \hat{y}_0 \\ 0 \end{bmatrix}.$$

11 Cones

We assume that we wish to fit a cone to m points (x_i, y_i, z_i), where $m \geq 6$. In the algorithm described below, we use a strategy similar to the one outlined in section 8. However, to deal with the case of cones with a wide apex angle, numerical considerations force us to adopt a slightly modified approach.

11.1 Parametrization

We specify a cone by

i) a point (x_0, y_0, z_0) on its axis,

ii) a vector (a, b, c) pointing along the axis,

iii) the angle ϕ at the apex, and

iv) information about where on the axis the cone is situated.

To parametrize a cone properly we need a systematic way of deciding which point on the axis to choose, along with a constraint on (a, b, c). Following section 8, for nearly vertical cones we set

$$c = 1, \tag{11.1}$$

while for the point on the axis we choose a constraint of the form

$$z_0 = s_0 - ax_0 - by_0, \tag{11.2}$$

for some constant s_0. We shall see in section 11.2 how to choose the constant s_0. Regarding iv), a first choice would be to use the cone vertex to specify position. However, for cones with small apex angle, the vertex could be situated very far from the data, leading to loss of numerical accuracy. A second choice is to specify the radius, i.e., the distance from the axis to the cone surface measured perpendicularly to the axis, at the point (x_0, y_0, z_0). This works well for cones with small apex angle, but not so well for cones with a wide angle. Our preferred choice is to specify the distance t from the point (x_0, y_0, z_0) to the cone surface, measured perpendicularly to cone surface, see Figure 11.1. We shall see that this parametrization also suffers from some defects for cones with a wide angle, but with care, these defects can be minimised.

11.2 Distance from a point to a cone

Given a cone specified by (x_0, y_0, z_0), (a, b, c), angle ϕ and distance t as above, the distance from a point (x_i, y_i, z_i) to the cone is found from

$$d_i = e_i \cos(\phi/2) + f_i \sin(\phi/2) - t, \tag{11.3}$$

where e_i is the distance from (x_i, y_i, z_i) to the line specified by (x_0, y_0, z_0) and (a, b, c); and f_i is the distance from (x_i, y_i, z_i) to the plane specified by (x_0, y_0, z_0) and (a, b, c). We have assumed that the vector (a, b, c) points along the axis in the direction of decreasing radius; if this is not so then we replace (a, b, c) by $-(a, b, c)$.

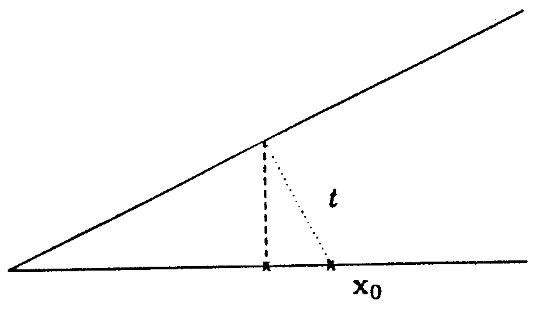

Figure 11.1: Parametrization of a cone.

If we employ the constraints (11.1) and (11.2), d_i is a function of the six parameters x_0, y_0, a, b, ϕ, and t, and to implement a Gauss-Newton algorithm to minimise the sum of squares of the distances we need to calculate the partial derivatives of d_i with respect to these six parameters. The expressions are algebraically complicated, but for the special case when $x_0 = y_0 = a = b = 0$, they simplify to

$$\begin{aligned} e_i &= r_i, \quad \text{where} \\ r_i &= \surd[x_i^2 + y_i^2]; \\ f_i &= z_i - s_0, \end{aligned} \tag{11.4}$$

and

$$\begin{aligned} \frac{\partial d_i}{\partial x_0} &= -x_i \cos(\phi/2)/r_i, \\ \frac{\partial d_i}{\partial y_0} &= -y_i \cos(\phi/2)/r_i, \\ \frac{\partial d_i}{\partial a} &= -x_i w_i/r_i, \\ \frac{\partial d_i}{\partial b} &= -y_i w_i/r_i, \\ \frac{\partial d_i}{\partial \phi} &= w_i/2, \\ \frac{\partial d_i}{\partial t} &= -1, \end{aligned} \tag{11.5}$$

with

$$w_i = (z_i - s_0)\cos(\phi/2) - r_i \sin(\phi/2). \tag{11.6}$$

We note that for cones with apex angle near π, the term $\cos(\phi/2)$ will be near zero and the x_0 and y_0 columns of the Jacobian will correspondingly be near zero. The Jacobian will then be nearly rank-deficient resulting in a degraded performance of the Gauss-Newton algorithm. In geometrical terms, the position of the vertex of the best-fit cone to nearly planar data is poorly defined - moving the vertex to any point on the plane has a small effect on the sum of the squares.

There remains the question of a suitable value for the constant s_0. We can show that in order to avoid unnecessary ill-conditioning in the Jacobian matrix, it is best to choose s_0 such that $\sum w_i = 0$, from which we get

$$s_0 = \bar{z} - \bar{r}\tan(\phi/2), \tag{11.7}$$

where $\bar{r}$ is the mean of the r_i. We can see that for cones with small apex angle and with s_0 given by (11.7), the point (x_0, y_0, z_0) will be close to the point on the axis nearest the centroid and if we assume that the centroid is at the origin we can safely set $s_0 = 0$. In the next section we describe an algorithm for this case. For wide-angled cones, (x_0, y_0, z_0) will be far from the data (and the cone surface) and this can give rise to numerical instability. For example, if we calculate d_i according to (11.3) then we risk cancellation errors. However, we can show that given an axis and apex angle the best-fit cone with this axis and angle contains the circle centred at the point on the axis nearest the centroid, with radius equal to the mean distance of the points to the axis. We can use this fact to produce a modified algorithm which obviates some of the numerical difficulties that occur for wide-angled cones.

11.3 Algorithm description

The algorithm to find the best-fit cone is based on the Gauss-Newton strategy outlined in sections 2.2 and 8. If the cone has a moderate apex angle (say $< 0.9\pi$) then, given estimates of i) a point (x_0, y_0, z_0) on its axis, ii) a vector (a, b, c) pointing along the axis, iii) the angle ϕ at the apex, and iv) the distance t from (x_0, y_0, z_0) to the cone surface, we can set $s_0 = 0$ and follow the steps:

I translate the data so that the point on the axis lies at the origin:

$$(x_i, y_i, z_i) := (x_i, y_i, z_i) - (x_0, y_0, z_0);$$

II transform the data by a rotation matrix U which rotates (a, b, c) to a point on the z-axis:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} := U \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix};$$

III form the right hand side vector $\mathbf{d}$ and Jacobian matrix according to (11.3) - (11.5) with $s_0 = 0$;

IV solve the linear least-squares system

$$J \begin{bmatrix} p_{x_0} \\ p_{y_0} \\ p_a \\ p_b \\ p_\phi \\ p_t \end{bmatrix} = -\mathbf{d};$$

V update the parameter estimates according to

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} := \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + U^T \begin{bmatrix} p_{x_0} \\ p_{y_0} \\ -p_{x_0}p_a - p_{y_0}p_b \end{bmatrix};$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} := U^T \begin{bmatrix} p_a \\ p_b \\ 1 \end{bmatrix};$$

$$\begin{aligned} \phi &:= \phi + p_\phi; \\ t &:= t + p_t. \end{aligned}$$

These steps are repeated until the algorithm has converged. In step I, we always start with (a copy of) the original data set rather than a transformed set from a previous iteration. If we wish to have (x_0, y_0, z_0) represent the point on the line nearest the origin, then we introduce one further step:

VI

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} := \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} - \frac{ax_0 + by_0 + cz_0}{a^2 + b^2 + c^2} \begin{bmatrix} a \\ b \\ c \end{bmatrix}.$$

We can adjust the above algorithm to ensure that we minimise any loss of numerical accuracy for wide-angled cones. For the modified algorithm we require that *the centroid of the data lies at the origin.* Given estimates of i) a point (x_0, y_0, z_0) on its axis, ii) a vector (a, b, c) pointing along the axis, and iii) the angle ϕ at the apex, one iteration follows the steps below:

I† calculate the point on the axis nearest the origin (= the centroid):

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} := \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} - \frac{ax_0 + by_0 + cz_0}{a^2 + b^2 + c^2} \begin{bmatrix} a \\ b \\ c \end{bmatrix};$$

II† translate the data so that the point on the axis lies at the origin:

$$(x_i, y_i, z_i) := (x_i, y_i, z_i) - (x_0, y_0, z_0);$$

III† transform the data by a rotation matrix U which rotates (a, b, c) to a point on the positive z-axis:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} := U \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix};$$

IV† calculate the mean distance of the points to the axis:

$$\begin{aligned} r_i &= \sqrt{[x_i^2 + y_i^2]}, \\ \bar{r} &= \frac{1}{m} \sum r_i; \end{aligned}$$

V† calculate the right hand side vector $\mathbf{d}$ according to

$$d_i = (r_i - \bar{r})\cos(\phi/2) + z_i \sin(\phi/2), \tag{11.8}$$

and Jacobian matrix according to (11.5) with

$$w_i = z_i \cos(\phi/2) - (r_i - \bar{r})\sin(\phi/2); \tag{11.9}$$

VI† solve the linear least-squares system

$$J \begin{bmatrix} p_{x_0} \\ p_{y_0} \\ p_a \\ p_b \\ p_\phi \\ p_t \end{bmatrix} = -\mathbf{d};$$

VII† update the parameter estimates according to

$$\begin{aligned} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} &:= \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + U^T \begin{bmatrix} p_{x_0} + \bar{r} p_a \tan(\phi/2) \\ p_{y_0} + \bar{r} p_b \tan(\phi/2) \\ -p_{x_0} p_a - p_{y_0} p_b \end{bmatrix}; \\ \begin{bmatrix} a \\ b \\ c \end{bmatrix} &:= U^T \begin{bmatrix} p_a \\ p_b \\ 1 \end{bmatrix}; \\ \phi &:= \phi + p_\phi. \end{aligned}$$

The modified algorithm implicitly has $s_0 = -\bar{r}\tan(\phi/2)$ (we have assumed that $\bar{z} = 0$) and so ensures that the Jacobian matrix is not unnecessarily ill-conditioned. Also at step V†, we use the fact that the best-fit cone with the given axis and angle contains the circle centred at the origin with radius $\bar{r}$, and this allows us to calculate the quantities d_i and w_i in a stable manner via (11.8) and (11.9). The modified algorithm is only slightly more complicated that the original algorithm and we recommend its use. However we can emphasise again that there will be ill-conditioning for near-planar cones, and that the computation will break down if $\phi = \pi$.

11.4 Notes

1. The situation for obtaining initial estimates of the parameters is much the same as that for cylinders (section 9.4). If we have estimates of (a, b, c) and ϕ, then we can rotate the data so that the axis is vertical and then fit

$$(x - \hat{x}_0)^2 + (y - \hat{y}_0)^2 = (r - kz)^2, \quad k = \tan(\phi/2),$$

to the x- and y-coordinates of the rotated points; this problem can be posed as a linear least-squares problem by a change of parameters similar to that employed for the circle (section 6.4). From the solution $\hat{x}_0$, $\hat{y}_0$ and r, we can derive estimates of (x_0, y_0, z_0), and subsequently t.

Acknowledgements
I am happy to thank my colleagues G. T. Anthony, H. M. Anthony and particularly M. G. Cox for their comments on drafts of this report, and for their many helpful suggestions.

References

Anthony, G. T., Anthony, H. M., Cox, M. G. and Forbes, A. B., 1991.
The parametrization of fundamental geometric form. Technical report, Commission of the European Communities - BCR, Luxembourg.

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

Anthony, G. T. and Cox, M. G., 1987.
The National Physical Laboratory's Data Approximation Subroutine Library. In *Algorithms for Approximation* (edited by J. C. Mason and M. G. Cox), pages 669 – 687. Clarendon Press, Oxford.

Boffey, T. B., Cox, M. G., Delves, L. M. and Pursglove, C. J., 1990.
Approximation by spheres. In *Algorithms for Approximation II* (edited by J. C. Mason and M. G. Cox). Chapman & Hall, London.

BS7172, 1989.
British Standard Guide to the Method of Assessment of Position, Size, and Departure from Nominal Form of Geometric Features. British Standards Institute, London.

Cox, M. G., 1981.
The least squares solution of overdetermined linear equations having band or augmented band structure. *IMA J. Numer. Anal.*, **1**, 3 – 22.

Cox, M. G., 1985.
Software for computer-aided measurement. *Laboratory Practice*, **34**, 59–63.

Cox, M. G. and Jones, H. M., 1989.
An algorithm for least-squares circle fitting to data with specified uncertainty ellipses. *IMA Journal of Numerical Analysis*, **9**, 285 – 298.

Dongarra, J. J. and Grosse, E., 1987.
Distribution of mathematical software via electronic mail. *Communications of the ACM.*, pages 403–407.

Dongarra, J. J., Moler, C. B., Bunch, J. R. and Stewart, G. W., 1979.
LINPACK Users' Guide. Society for Industrial and Applied Mathematics, Philadelphia.

Fletcher, R., 1980.
Practical Optimization, Vol. 1. John Wiley and Sons, Chichester.

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

Forbes, A. B., 1990.
Least squares best fit geometric elements. In *Algorithms for Approximation* (edited by J. C. Mason and M. G. Cox). Chapman & Hall, London.

Forbes, A. B., 1991.
Surface fitting in coordinate metrology. DITC report, National Physical Laboratory, Teddington. In preparation.

Ford, B., Bentley, J., Du Croz, J. J. and Hague, S. J., 1979.
The NAG Library 'machine'. *Software - Practice and Experience*, **9**, 56–72.

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

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

Hodson, S. M., Jones, H. M. and Long, E. M. R., 1983.
A brief guide to the NPL Numerical Optimization Software Library. Technical report, National Physical Laboratory, Teddington.

149/89	MANSFIELD, A.J.	An explanation of the formal definition of Prolog	Oct 1989
150/89	O'Neill, G.	Rapid prototyping of formal specifications using Miranda	Nov 1989
151/89	PRICE, W.L.	Hash functions - a tutorial and status report	Nov 1989
152/89	WARHAM, A.G.P.	Annular slot antenna radiating into lossy material	Nov 1989
153/89	FORBES, A.B.	Robust circle and sphere fitting by least squares	Nov 1989
154/89	HARDING, C. and RENGGER, R.	The first steps towards a Usability Problem Description Language	Nov 1989
155/89	FORBES, A.B. and MANSFIELD, A.J.	Neural implementation of a method for solving systems of linear algebraic equations	Nov 1989
156/90	JANSEN, B.	Storage allocation in Ada	Feb 1990
157/90	COX, M.G.	Mathematical modelling in manufacturing metrology	Feb 1990
158/90	PRETOR-PINNEY, G.E. and RENGGER, R.E.	Criteria for device selection: A comparison between a mouse and a touchscreen as an input device for interactive video	Feb 1990
159/90	MAISSEL, J.	Development of a methodology for icon evaluation	Jan 1990
160/90	PARKIN, G.I. and O'NEILL, G.	Specification of the MAA Standard in VDM	Feb 1990
161/90	NORTH, N.D.	Automatic test generation for the triangle problem	Feb 1990
162/90	NORTH, N.D.	An implementation of sets and maps as Miranda abstract data types	Feb 1990

163/90	RENGGER, R.E. A preliminary design for a methodology for Experimentally Measuring Usability - EMU	Apr 1990
164/90	BROCKLEHURST, E.R. and POBGEE, P.J. Parallel processing, a glossary of terms	May 1990
165/90	PITT, D., BOSHIER, A.G. and SZCZYGIEL, B.M. One2One - A tool for translating ASN.1 to ACT ONE	Jun 1990
166/90	COX, M.G. Algorithms for spline curves and surfaces	Jun 1990
167/90	WICHMANN, B.A. Getting the correct answer	Jun 1990
168/90	LONG, E.M.R. Algorithms for the solution of constrained non-linear least squares problems	Aug 1990
169/90	WONG, G-K. and RENGGER, R. The validity of questionnaires designed to measure user-satisfaction of computer systems	Oct 1990
170/90	BROCKLEHURST, E.R. and POBGEE, P.J. Parallel processing, a glossary of terms (Update)	Nov 1990
171/91	DICKINSON, I.P. A VDM-SL specification for the 'vi' pattern matcher	Jan 1991