[J. Res. Natl. Inst. Stand. Technol. **103**, 633 (1998)]

Least-Squares Fitting Algorithms of the NIST Algorithm Testing System

Volume 103 Number 6 November–December 1998

Craig M. Shakarji

National Institute of Standards and Technology,
Gaithersburg, MD 20899-0001

This report describes algorithms for fitting certain curves and surfaces to points in three dimensions. All fits are based on orthogonal distance regression. The algorithms were developed as reference software for the National Institute of Standards and Technology's Algorithm Testing System, which has been used for 5 years by NIST and by members of the American Society of Mechanical Engineers' B89.4.10 standards committee. The Algorithm Testing System itself is described only briefly; the main part of this paper covers the general linear algebra, numerical analysis, and optimization methods it employs. Most of the fitting routines rely on the Levenberg-Marquardt optimization routine.

Key words: coordinate measuring machine; curve fitting; least-squares fitting; Levenberg-Marquardt; orthogonal distance regression; surface fitting.

Accepted: August 7, 1998

Available online: http://www.nist.gov/jres

1. Introduction

Mathematical software, particularly curve and surface fitting routines, is a critical component of coordinate metrology systems. An important but difficult task is to assess the performance of such routines [1,2]. The National Institute of Standards and Technology has developed a software package, the NIST Algorithm Testing System (ATS), that can assist in assessing the performance of geometry-fitting routines [3]. This system has been aiding in the development of a U. S. standard (American Society of Mechanical Engineers (ASME) B89.4.10) for software performance evaluation of coordinate measuring systems [4]. The ATS is also the basis of NIST's Algorithm Testing and Evaluation Program for Coordinate Measuring Systems (ATEP-CMS) [5,6].

The ATS incorporates three core modules: a *data generator* [7] for defining and generating test data sets, a collection of *reference algorithms* which provides a performance baseline for fitting algorithms, and a *comparator* for analyzing the results of fitting algorithms versus the reference algorithms. This paper concentrates on the development of highly accurate reference algorithms.

The paper is organized as follows. We first introduce notation and certain key functions and derivatives that will be used throughout the paper. Section 2 describes fitting algorithms for linear geometries—planes and lines. Lagrange multipliers [8] are used in solving the constrained minimization problems, which are developed into standard eigenvector problems. Section 3 deals with nonlinear geometry fitting. We use an unconstrained optimization method that requires derivatives of appropriate distance functions. These required functions and derivatives are provided for the reader. Appendix A gives an outline of the unconstrained optimization algorithm (Levenberg-Marquardt) that is modified to allow for normalization of fitting parameters within the routine. Appendix B gives, for all the geometries, the appropriate derivatives needed to create a valuable check for a local minimum.

1.1 Notation and Preliminary Remarks

Assume we are fitting a set of data points, $\{\boldsymbol{x}_i\}$, $i = 1, 2, ..., N$, that have been translated so that their centroid is the origin. Usually scalar quantities are represented in plain type and matrix or vector quantities with boldface type. Other notation:

$\boldsymbol{x} = (x, y, z)$	A point in 3-dimensional space.
$\lvert\cdot\rvert$	The Euclidean ($\boldsymbol{L}_2$) norm. E. g., $\lvert\boldsymbol{x}\rvert = \sqrt{x^2 + y^2 + z^2}$.
$\boldsymbol{x}_i = (x_i, y_i, z_i)$	The ith data point.
$\bar{\boldsymbol{x}} = (\bar{x}, \bar{y}, \bar{z})$	The centroid of the data, $\frac{1}{N}\left(\sum x_i, \sum y_i, \sum z_i\right)$. (Note: These and all other sums in this paper are taken from $i = 1, 2, ..., N$.)
$\boldsymbol{A} = (A, B, C)$	Direction numbers that specify an orientation, $\boldsymbol{A} \neq 0$.
$\boldsymbol{a} = (a, b, c)$	Direction cosines that specify an orientation. Note: $\lvert\boldsymbol{a}\rvert = 1$. An orientation's direction numbers can be converted into direction cosines by: $\boldsymbol{a} = \boldsymbol{A}/\lvert\boldsymbol{A}\rvert$.
J	The objective function. J is the sum of the squares of the distances from the data points to the geometry. $J = \sum d_i^2$.
$\boldsymbol{M}$	The $N \times 3$ matrix containing the data points: $\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix}$
∇	The gradient of a scalar function. E.g., $\nabla h(x, y, z) = (h_x, h_y, h_z) = \left(\frac{\partial h}{\partial x}, \frac{\partial h}{\partial y}, \frac{\partial h}{\partial z}\right)$.

For each geometry, we show the defining parameters, the equation for the orthogonal distance from a single data point to the geometry, the objective function, and a brief description of the steps in the calculation.

2. Linear Geometries

Linear geometries (lines and planes) are solved using Lagrange multipliers on a constrained minimization problem. Both cases reduce to a rather simple eigen-problem.

2.1 Plane Fitting

Defining parameters:
$\boldsymbol{x}$—a point on the plane.
$\boldsymbol{a}$—the direction cosines of the normal to the plane.

Distance equation:
$d_i = d(\boldsymbol{x}_i) = d(\boldsymbol{x}_i, \boldsymbol{x}, \boldsymbol{a}) = \boldsymbol{a} \cdot (\boldsymbol{x}_i - \boldsymbol{x})$

Objective function:
$J(\boldsymbol{x}, \boldsymbol{a}) = \sum [\boldsymbol{a} \cdot (\boldsymbol{x}_i - \boldsymbol{x})]^2$

Description:
The centroid of the data must lie on the least-squares plane. This can be seen because $\nabla J = 0$ at the least squares solution, yielding $\sum \boldsymbol{a} \cdot (\boldsymbol{x}_i - \boldsymbol{x}) = 0$. Multiplying by $1/N$ gives $\frac{a}{N}\sum(x_i - x) + \frac{b}{N}\sum(y_i - y) + \frac{c}{N}\sum(z_i - z) = 0$. Distributing the summation gives $a(\bar{x} - x) + b(\bar{y} - y) + c(\bar{z} - z) = 0$, which is to say $d(\bar{\boldsymbol{x}}, \boldsymbol{x}, \boldsymbol{A}) = 0$, i.e., $\bar{\boldsymbol{x}}$ lies on the least-squares plane. Since by assumption the data points have been translated to the origin, and since the centroid of the data must be a point on the least squares plane, we can set $\boldsymbol{x} = 0$.

The direction of the fitted plane, $\boldsymbol{a}$, can be found by solving the constrained minimization problem, namely, minimizing J subject to the constraint that $\lvert\boldsymbol{a}\rvert = 1$. Define a function, $G = \lvert\boldsymbol{a}\rvert^2 - 1$, so that the problem is to minimize J subject to the constraint that $G = 0$. The method of Lagrange multipliers [8] tells us that the minimum occurs at a point where $\nabla J = \lambda \nabla G$, for some real number λ. (Here, a, b, and c are treated as independent variables, since the constraint is accounted for in G. Therefore, $\nabla = (\partial/\partial a, \partial/\partial b, \partial/\partial c)$.) But $\nabla G = 2\boldsymbol{a}$, and $\nabla J = 2(\boldsymbol{M}^{\mathsf{T}}\boldsymbol{M})\boldsymbol{a}$, yielding the eigen-problem, $(\boldsymbol{M}^{\mathsf{T}}\boldsymbol{M})\boldsymbol{a} = \lambda\boldsymbol{a}$, referred to as the *normal equations*.

This 3×3 eigenvector problem can be easily solved using well-established routines (e.g., Jacobi iterations [9]). However, we note that the eigenvectors of $\boldsymbol{M}^{\mathsf{T}}\boldsymbol{M}$ are also the singular vectors (from the singular value decomposition) of $\boldsymbol{M}$ [9]. This allows us to gain numerical stability by applying the singular value decomposition (SVD) to $\boldsymbol{M}$ without ever computing $\boldsymbol{M}^{\mathsf{T}}\boldsymbol{M}$, which is the method implemented in the ATS.

Finally, we must determine how to select the correct eigenvector (i.e., singular vector) of the three produced by the SVD. The normal equations can be written as follows:

$$\sum x_i(\boldsymbol{a} \cdot \boldsymbol{x}_i) = \lambda a$$
$$\sum y_i(\boldsymbol{a} \cdot \boldsymbol{x}_i) = \lambda b$$
$$\sum z_i(\boldsymbol{a} \cdot \boldsymbol{x}_i) = \lambda c$$

Multiplying these three equations by a, b, and c, respectively, then summing the equations gives $\sum(\boldsymbol{a}\cdot\boldsymbol{x}_i)^2 = \lambda|\boldsymbol{a}|^2 = \lambda$.

But the sum on the left is just the objective function, i.e., the sum of the squares of the residuals. Thus the value of λ is the objective function, hence the correct eigenvector for the least-squares solution corresponds to the smallest eigenvalue. When using the SVD, we choose the singular vector corresponding to the smallest singular value, since under these conditions the singular values are the square roots of the eigenvalues [10].

2.2 Line Fitting

Defining parameters:
$\boldsymbol{x}$—a point on the line.
$\boldsymbol{a}$—the direction cosines of the line.

Distance equation:
$d(\boldsymbol{x}_i) = d(\boldsymbol{x}_i,\boldsymbol{x},\boldsymbol{a}) = |\boldsymbol{a}\times(\boldsymbol{x}_i-\boldsymbol{x})|$ (which also equals $\sqrt{|\boldsymbol{x}_i-\boldsymbol{x}|^2-[\boldsymbol{a}\cdot(\boldsymbol{x}_i-\boldsymbol{x})]^2}$) where $\times$ denotes the vector cross product.

Objective function:
$J(\boldsymbol{x},\boldsymbol{a}) = \sum|\boldsymbol{a}\times(\boldsymbol{x}_i-\boldsymbol{x})|^2$

Description:
Since the data has been translated to the origin, and since the centroid of the data must be a point on the least squares plane (similar to the case above with the least squares plane), we set $\boldsymbol{x} = 0$.

The direction $\boldsymbol{a}$ can be found by following the same strategy as in the case of a plane. For line fitting, the normal equations are $\boldsymbol{M}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{a} = \lambda\boldsymbol{a}$ just as in the case of plane fitting. Once again, the correct eigenvector must be chosen to minimize the sum-of-squares of the residuals. As shown with planes, we obtain $\lambda = \sum(\boldsymbol{a}\cdot\boldsymbol{x})^2$, so $J = -\lambda + \sum|\boldsymbol{x}_i|^2$, meaning that J is minimized when λ is maximized. Thus the correct eigenvector choice is the one corresponding to the largest eigenvalue. As in the case of plane fitting, numerical stability is gained by finding the eigenvectors of $\boldsymbol{M}$ through the SVD, rather than by solving the normal equations. Since the singular values are the square roots of the eigenvalues [10], we choose the eigenvector corresponding to the largest singular value.

3. Nonlinear Geometries

3.1 Utility Functions f and g

The line and plane distance functions arise quite often in this paper, thus we define them here, calling them f and g respectively, giving necessary derivatives, which are used throughout the rest of this paper. We compute the nonlinear fits using unconstrained minimization algorithms, so we define the line and plane distance functions in terms of direction numbers rather than direction cosines.

Let $g(\boldsymbol{x}_i,\boldsymbol{x},\boldsymbol{A})$ denote the distance from the point, $\boldsymbol{x}_i$, to the plane defined by the point, $\boldsymbol{x}$, and the normal direction, $\boldsymbol{a} = \boldsymbol{A}/|\boldsymbol{A}|$. The value of g is given by: $g_i = g(\boldsymbol{x}_i,\boldsymbol{x},\boldsymbol{A}) = \boldsymbol{a}\cdot(\boldsymbol{x}_i-\boldsymbol{x}) = a(x_i-x)+b(y_i-y)+c(z_i-z)$.

Let $f(\boldsymbol{x}_i,\boldsymbol{x},\boldsymbol{A})$ denote the distance from the point, $\boldsymbol{x}_i$, to the line defined by the point, $\boldsymbol{x}$, and the direction, $\boldsymbol{a} = \boldsymbol{A}/|\boldsymbol{A}|$. The value of f is given by: $f_i = f(\boldsymbol{x}_i,\boldsymbol{x},\boldsymbol{A}) = |\boldsymbol{a}\times(\boldsymbol{x}_i-\boldsymbol{x})|$. That is,

$$f_i = \sqrt{u^2+v^2+w^2},\ \text{where}\ \begin{aligned} u &= c(y_i-y)-b(z_i-z)\\ v &= a(z_i-z)-c(x_i-x)\\ w &= b(x_i-x)-a(y_i-y)\end{aligned}$$

This expression for f is used because of its numerical stability. One should note that f could also be expressed (for derivative calculations) as $f_i = \sqrt{|\boldsymbol{x}_i-\boldsymbol{x}|^2-g_i^2}$.

Note: A, B, and C are independent variables, whereas a, b, and c are not, because the constraint $a^2+b^2+c^2=1$ causes a, b, and c to depend on each other. When dealing with the nonlinear geometries we treat f and g as functions dependent on $\boldsymbol{A}$, as opposed to $\boldsymbol{a}$, in order to use unconstrained minimization algorithms. Treating f and g as functions dependent on $\boldsymbol{a}$ would force us to restrict ourselves to using constrained minimization solvers. In the linear cases, we *did* solve constrained minimization problems. So when we differentiate with respect to A, for example, we treat a, b, and c all as functions of A, B, and C (e.g., $a = A/\sqrt{A^2+B^2+C^2}$). This yields the following array of derivatives:

$$\begin{bmatrix}\nabla a\\ \nabla b\\ \nabla c\end{bmatrix} = \begin{bmatrix}\frac{\partial a}{\partial A} & \frac{\partial a}{\partial B} & \frac{\partial a}{\partial C}\\ \frac{\partial b}{\partial A} & \frac{\partial b}{\partial B} & \frac{\partial b}{\partial C}\\ \frac{\partial c}{\partial A} & \frac{\partial c}{\partial B} & \frac{\partial c}{\partial C}\end{bmatrix} = \frac{1}{|\boldsymbol{A}|}\begin{bmatrix}1-a^2 & -ab & -ac\\ -ab & 1-b^2 & -bc\\ -ac & -bc & 1-c^2\end{bmatrix}$$

These algorithms normalize $\boldsymbol{A}$ at every step, so for simplicity of expressing derivatives, assume $|\boldsymbol{A}| = 1$. $\boldsymbol{A}$ remains unconstrained; we just assume it happens to have unit magnitude.

The derivatives for f_i and g_i are then

$$\frac{\partial g_i}{\partial x} = -a$$
$$\frac{\partial g_i}{\partial y} = -b$$
$$\frac{\partial g_i}{\partial z} = -c$$
$$\frac{\partial g_i}{\partial A} = (x_i - x) - ag_i$$
$$\frac{\partial g_i}{\partial B} = (y_i - y) - bg_i$$
$$\frac{\partial g_i}{\partial C} = (z_i - z) - cg_i$$
$$\frac{\partial f_i}{\partial x} = [ag_i - (x_i - x)]/f_i$$
$$\frac{\partial f_i}{\partial y} = [bg_i - (y_i - y)]/f_i$$
$$\frac{\partial f_i}{\partial z} = [cg_i - (z_i - z)]/f_i$$
$$\frac{\partial f_i}{\partial A} = g_i[ag_i - (x_i - x)]/f_i$$
$$\frac{\partial f_i}{\partial B} = g_i[bg_i - (y_i - y)]/f_i$$
$$\frac{\partial f_i}{\partial C} = g_i[cg_i - (z_i - z)]/f_i$$

The above derivatives of f_i are undefined when $f_i = 0$ (i.e., when $\boldsymbol{x}_i$ is on the line.) In this rarely needed case the gradient is given by:

$$\left(\sqrt{1-a^2}, \sqrt{1-b^2}, \sqrt{1-c^2},\right.$$
$$\left. g\sqrt{1-a^2}, g\sqrt{1-b^2}, g\sqrt{1-c^2}\right).$$

For cylinders, cones, and tori, the line associated with f is the geometry's axis. For cones the plane associated with g is the plane through the point, $\boldsymbol{x}$, perpendicular to the axis. For tori, the plane associated with $\boldsymbol{g}$ is the plane perpendicular to the axis that divides the torus in half.

3.2 Choice of Algorithm

Good optimization algorithms can readily be found to minimize the objective function, J. Usually such an algorithm will require an initial guess, along with partial derivatives, either of J itself or of the distance function, d_i. Both sets of derivatives are given in this paper (those for J in the appendix). These should enable a reader to implement a least-squares algorithm even if the optimization algorithm used differs from the author's choice, which follows.

In the ATS, nonlinear geometries are fit in an unconstrained manner using the Levenberg-Marquardt algorithm. The algorithm requires an initial guess as well as the first derivatives of the distance function. In practice it converges quickly and accurately even with a wide range of initial guesses. Details of this algorithm are given in appendix A. Additionally, the code allows us to normalize the fitting parameters after every iteration of the algorithm. For each geometry we list the distance and objective functions, the appropriate derivatives, and the parameter normalization we use.

3.3 Sphere Fitting

Defining parameters:
$\boldsymbol{x}$—the center of the sphere.
r—the radius of the sphere.

Distance equation:
$d(\boldsymbol{x}_i) = |\boldsymbol{x}_i - \boldsymbol{x}| - r$

Objective function:
$J(\boldsymbol{x}, r) = \sum(|\boldsymbol{x}_i - \boldsymbol{x}| - r)^2$

Normalization:
(None)

Derivatives:

$$\frac{\partial d_i}{\partial x} = -(x_i - x)/|\boldsymbol{x}_i - \boldsymbol{x}|$$
$$\frac{\partial d_i}{\partial y} = -(y_i - y)/|\boldsymbol{x}_i - \boldsymbol{x}|$$
$$\frac{\partial d_i}{\partial z} = -(z_i - z)/|\boldsymbol{x}_i - \boldsymbol{x}|$$
$$\frac{\partial d_i}{\partial r} = -1$$

3.4 Two-Dimensional Circle Fitting

This case is simply the sphere fit (above) restricted to two-dimensions:

Defining parameters:
x—the x-coordinate of the center of the circle.
y—the y-coordinate of the center of the circle.
r—the radius of the circle.

Distance equation:
$d(x_i, y_i) = \sqrt{(x_i - x)^2 + (y_i - y)^2} - r$

Objective function:
$$J(x, y, r) = \sum\left(\sqrt{(x_i - x)^2 + (y_i - y)^2} - r\right)^2$$

Normalization:
(None)

Derivatives:

$$\frac{\partial d_i}{\partial x} = -(x_i - x)/(d_i + r)$$
$$\frac{\partial d_i}{\partial y} = -(y_i - y)/(d_i + r)$$
$$\frac{\partial d_i}{\partial r} = -1$$

3.5 Three-Dimensional Circle Fitting

Defining parameters:
x—the center of the circle.
A—the direction numbers of the normal to circle's plane.
r—the radius of the circle.

Distance equation:

$$d(\boldsymbol{x}_i) = \sqrt{g_i^2 + (f_i - r)^2}$$

Objective function:

$$J(\boldsymbol{x}, \boldsymbol{A}, r) = \sum \left(g_i^1 + (f_i - r)^2 \right)$$

Normalization:
$\boldsymbol{A} \leftarrow \boldsymbol{A}/|\boldsymbol{A}|$ (Here and elsewhere, "←" denotes assignment of value. In this case, the value of **A** is replaced by the value $\boldsymbol{A}/|\boldsymbol{A}|$.)

Derivatives:

$$\frac{\partial d_i}{\partial x} = [g_i(g_i)_x + f_i(f_i)_x]/d_i$$
$$\frac{\partial d_i}{\partial y} = [g_i(g_i)_y + f_i(f_i)_y]/d_i$$
$$\frac{\partial d_i}{\partial z} = [g_i(g_i)_z + f_i(f_i)_z]/d_i$$
$$\frac{\partial d_i}{\partial A} = [g_i(g_i)_A + f_i(f_i)_A]/d_i$$
$$\frac{\partial d_i}{\partial B} = [g_i(g_i)_B + f_i(f_i)_B]/d_i$$
$$\frac{\partial d_i}{\partial C} = [g_i(g_i)_C + f_i(f_i)_C]/d_i$$
$$\frac{\partial d_i}{\partial r} = -(f_i - r)/d_i$$

Description:
We use a multi-step process to accomplish 3D circle fitting:

1. Compute the least-squares plane of the data.
2. Rotate the data such that the least-squares plane is the x-y plane.
3. Project the rotated data points onto the x-y plane.
4. Compute the 2D circle fit in the x-y plane.
5. Rotate back to the original orientation.
6. Perform a full 3D minimization search over all the parameters.

Some coordinate measuring system software packages stop at step (5) and report the orientation, center, and radius as the least-squares circle in 3D. This approach is valid when the projection onto the plane is done simply to compensate for measurement errors on points which would otherwise be coplanar. But this method does not in general produce the circle yielding the least sum-of-squares possible (even though it is usually a good approximation.) In order to achieve the true 3D least-squares fit, the circle computed at step (5) is used as an initial guess in the Levenberg-Marquardt algorithm, which optimizes over all the parameters simultaneously [step (6)].

Step (2) is carried out using the appropriate rotation matrix to rotate the direction, **a**, to the z-direction, namely,

$$\begin{bmatrix} 1-\frac{a^2}{1+c} & \frac{-ab}{1+c} & -a \\ \frac{-ab}{1+c} & 1-\frac{b^2}{1+c} & -b \\ a & b & c \end{bmatrix}$$

If $c < 0$, **a** is replaced with $-\boldsymbol{a}$, thus rotating the direction to the minus z-direction (which is adequate for our purposes.) Step (5) is carried out using the appropriate rotation matrix to rotate the z-direction to the direction, **a**. Namely,

$$\begin{bmatrix} 1-\frac{a^2}{1+c} & \frac{-ab}{1+c} & a \\ \frac{-ab}{1+c} & 1-\frac{b^2}{1+c} & b \\ -a & -b & c \end{bmatrix}$$

3.6 Cylinder Fitting

Defining parameters:
x—a point on the cylinder axis.
A—the direction numbers of the cylinder axis.
r—the radius of the cylinder.

Distance equation:
$d(\boldsymbol{x}_i) = f_i - r$

Objective function:
$J(\boldsymbol{x}, A, r) = \sum (f_i - r)^2$

Normalization:
$\boldsymbol{A} \leftarrow \boldsymbol{A}/|\boldsymbol{A}|$
$\boldsymbol{x} \leftarrow$ (point on axis closest to origin)

Derivatives:
$\frac{\partial d_i}{\partial x} = (f_i)_x$
$\frac{\partial d_i}{\partial y} = (f_i)_y$
$\frac{\partial d_i}{\partial z} = (f_i)_z$
$\frac{\partial d_i}{\partial A} = (f_i)_A$
$\frac{\partial d_i}{\partial B} = (f_i)_B$
$\frac{\partial d_i}{\partial C} = (f_i)_C$
$\frac{\partial d_i}{\partial r} = -1$

3.7 Cone Fitting

Defining parameters:
$\boldsymbol{x}$—a point on the cone axis (not the apex).
$\boldsymbol{A}$—the direction numbers of the cone axis (pointing toward the apex).
s—the orthogonal distance from the point, $\boldsymbol{x}$, to the cone.
ψ—the cone's apex semi-angle.

Distance equation:
$d(\boldsymbol{x}_i) = f_i \cos\psi + g_i \sin\psi - s$

Objective function:
$J(\boldsymbol{x}, \boldsymbol{A}, s, \psi) = \sum (f_i \cos\psi + g_i \sin\psi - s)^2$

Normalization:
$\boldsymbol{A} \leftarrow \boldsymbol{A}/|\boldsymbol{A}|$
$x \leftarrow$ (point on axis closest to origin)
$\psi \leftarrow \psi (\mathrm{mod}\ 2\pi)$
if $\psi > \pi$ then $[\psi \leftarrow \psi (\mathrm{mod}\ \pi);\ \boldsymbol{A} \leftarrow -\boldsymbol{A}]$
if $\psi > \frac{\pi}{2}$ then $\psi \leftarrow \pi - \psi$
if $s < 0$ then $[s \leftarrow -s;\ \boldsymbol{A} \leftarrow -\boldsymbol{A}]$

Derivatives:
$\frac{\partial d_i}{\partial x} = (f_i)_x \cos\psi + (g_i)_x \sin\psi$
$\frac{\partial d_i}{\partial y} = (f_i)_y \cos\psi + (g_i)_y \sin\psi$
$\frac{\partial d_i}{\partial z} = (f_i)_z \cos\psi + (g_i)_z \sin\psi$
$\frac{\partial d_i}{\partial A} = (f_i)_A \cos\psi + (g_i)_A \sin\psi$
$\frac{\partial d_i}{\partial B} = (f_i)_x \cos\psi + (g_i)_x \sin\psi$
$\frac{\partial d_i}{\partial C} = (f_i)_x \cos\psi + (g_i)_x \sin\psi$
$\frac{\partial d_i}{\partial s} = -1$
$\frac{\partial d_i}{\partial \psi} = -f_i \sin\psi + g_i \cos\psi$

3.8 Torus Fitting

Defining parameters:
$\boldsymbol{x}$—the torus center.
$\boldsymbol{A}$—the direction numbers of the torus axis.
r—the major radius.
R—the minor radius.

Distance equation:
$d(\boldsymbol{x}_i) = \sqrt{g_i^2 + (f_i - r)^2} - R$

Objective function:
$$J(\boldsymbol{x}, \boldsymbol{A}, r, R) = \sum \left[\sqrt{g_i^2 + (f_i - r)^2} - R\right]^2$$

Normalization:
$A \leftarrow \boldsymbol{A}/|\boldsymbol{A}|$

Derivatives:
$\frac{\partial d_i}{\partial x} = [g_i (g_i)_x + (f_i - r)(f_i)_x]/(d_i + R)$
$\frac{\partial d_i}{\partial y} = [g_i (g_i)_y + (f_i - r)(f_i)_y]/(d_i + R)$
$\frac{\partial d_i}{\partial z} = [g_i (g_i)_z + (f_i - r)(f_i)_z]/(d_i + R)$
$\frac{\partial d_i}{\partial A} = [g_i (g_i)_A + (f_i - r)(f_i)_A]/(d_i + R)$
$\frac{\partial d_i}{\partial B} = [g_i (g_i)_B + (f_i - r)(f_i)_B]/(d_i + R)$
$\frac{\partial d_i}{\partial C} = [g_i (g_i)_C + (f_i - r)(f_i)_C]/(d_i + R)$
$\frac{\partial d_i}{\partial r} = -(f_i - r)(d_i + R)$
$\frac{\partial d_i}{\partial R} = -1$

4. Discussion

The algorithms have been implemented in the ATS and have been used for 5 years by NIST and by members of the ASME B89.4.10 Working Group. In general they have performed extremely well. They have successfully solved a number of difficult fitting problems that could not be solved by many commercial software packages used on Coordinate Measuring Systems (CMSs). (Some of the most difficult problems are cylinders or cones sampled over a small patch.) The ATS algorithms have an extremely broad range of convergence. Failure to converge has only been observed for pathological fitting problems (e.g., fitting a circle to collinear points). Special checks can detect most of these situations.

The ATS algorithms are generally robust. For most fits, a good starting guess is not required to reach the global minimum. This is due, in part, to the careful choice of fitting parameters, the use of certain constraints, and, for cylinders and cones, the technique of restarting a search after an initial solution is found.

5. Appendix A. The Levenberg-Marquardt Algorithm

The Levenberg-Marquardt algorithm [11] finds the vector, $\boldsymbol{p}$, that minimizes an objective function, $J(\boldsymbol{p}) = \Sigma d_i^2(\boldsymbol{p})$, where $d_i(\boldsymbol{p})$ is the distance from the point, $\boldsymbol{x}_i$, to the geometry defined by the vector of parameters, $\boldsymbol{p}$. In the case of a cylinder, for example, $\boldsymbol{p} = (\boldsymbol{x}, \boldsymbol{A}, r) = (x, y, z, A, B, C, r)$ and $d_i(\boldsymbol{p}) = f_i - r$. One derivation of the algorithm [12] begins by approximating J as a linear function of $\boldsymbol{p}$, $\hat{J}(\boldsymbol{p})$:

$$J(\boldsymbol{p}) \approx \hat{J}(\boldsymbol{p}) = \sum (d_i(\boldsymbol{p}_0) + \nabla d_i(\boldsymbol{p}_0) \cdot \boldsymbol{p})^2$$

where $\nabla d_i(\boldsymbol{p}_0)$ is the gradient of $d_i(\boldsymbol{p})$ evaluated at an initial guess, $\boldsymbol{p}_0$. This approximation is valid within a certain trust region radius. The derivation then considers how to minimize $\hat{J}(\boldsymbol{p})$. A search direction is calculated based on $\hat{J}(\boldsymbol{p})$, and a search is made in that direction within the limits of the trust region radius for a point, $\boldsymbol{p}_{\text{new}}$, such that $J(\boldsymbol{p}_{\text{new}}) < J(\boldsymbol{p}_0)$. When $\boldsymbol{p}_{\text{new}}$ is found, it becomes the new $\boldsymbol{p}_0$ for another iteration of the above process.

The basis for the algorithm is the result that the solution, $\boldsymbol{p}^*$, to each iteration can be expressed as $\boldsymbol{p}^* = \boldsymbol{p}(\lambda) = -(\boldsymbol{F}_0^{\mathsf{T}}\boldsymbol{F}_0 + \lambda \boldsymbol{D}^{\mathsf{T}}\boldsymbol{D})^{-1}\boldsymbol{F}_0^{\mathsf{T}}\boldsymbol{d}(\boldsymbol{p}_0)$ where $\boldsymbol{F}_0$ is a matrix having $\nabla d_i(\boldsymbol{p}_0)$ as its ith row, $\boldsymbol{D}$ is an appropriate weighting matrix, $\boldsymbol{d}(\boldsymbol{p}_0)$ is the vector of residuals, $d_i(\boldsymbol{p}_0)$, and $\lambda \geq 0$ is a variable called the Levenberg-Marquardt parameter. This parameter can be considered the Lagrange multiplier for the constraint that each search be limited to the trust region radius.

The Levenberg-Marquardt algorithm is presented in the figure below. The matrix $\boldsymbol{F}_0^{\mathsf{T}}\boldsymbol{F}_0 + \lambda \boldsymbol{D}^{\mathsf{T}}\boldsymbol{D}$ is named $\boldsymbol{H}$ and the vector $\boldsymbol{F}_0^{\mathsf{T}}\boldsymbol{d}(\boldsymbol{p}_0)$ is called $\boldsymbol{v}$. The algorithm includes a modification suggested by Nash [11], which is to use a weighting matrix defined so that $\boldsymbol{D}^{\mathsf{T}}\boldsymbol{D}$ is the identity matrix plus the diagonal of $\boldsymbol{F}_0^{\mathsf{T}}\boldsymbol{F}_0$. Nash's use of

Input is an initial guess vector, $\boldsymbol{p}_0$, containing the defining parameters. Returns $\boldsymbol{p}_0$ that minimizes J.

```
Set $\lambda \leftarrow 0.0001$
Repeat
    Decrement $\lambda$;  Normalize $\boldsymbol{p}_0$
    Set $\boldsymbol{U} \leftarrow \boldsymbol{F}_0^{\mathsf{T}}\boldsymbol{F}$;   $\boldsymbol{v} \leftarrow \boldsymbol{F}_0^{\mathsf{T}}\boldsymbol{d}(\boldsymbol{p}_0)$;   $J_0 \leftarrow \sum d_i^2(\boldsymbol{p}_0)$
    Repeat
        Increment $\lambda$
        Set $\boldsymbol{H} \leftarrow \boldsymbol{U} + \lambda(\boldsymbol{I} + \mathrm{diag}(u_{11}, u_{22}, \cdots, u_{nn}))$
        Solve the system $\boldsymbol{Hx} = -\boldsymbol{v}$
        Set $\boldsymbol{p}_{\text{new}} \leftarrow \boldsymbol{p}_0 + \boldsymbol{x}$;   $J_{\text{new}} \leftarrow \sum d_i^2(\boldsymbol{p}_{\text{new}})$
        If converged, Set $\boldsymbol{p}_0 \leftarrow$ Normalized $\boldsymbol{p}_{\text{new}}$; return $\boldsymbol{p}_0$
    Until $J_{\text{new}} < J_0$ or an iteration limit is reached
    If $J_{\text{new}} < J_0$,  $\boldsymbol{p}_0 \leftarrow \boldsymbol{p}_{\text{new}}$
Until an iteration limit is reached
```

Fig. 1. Levenberg-Marquardt algorithm.

the identity in the definition of ***D*** forces ***H*** to be positive definite. (Note that ***D*** is never calculated.) Since ***H*** is also symmetric, the system $\boldsymbol{Hx} = -\boldsymbol{v}$ can be reliably solved using the Cholesky decomposition [9]. We chose 10 and 0.04 as factors with which to increment and decrement λ respectively. Finally, note that we normalize the parameter vector, $\boldsymbol{p}$, at every iteration. However, normalization of $\boldsymbol{p}$ never changes the value of the objective function, $J(\boldsymbol{p})$. This routine is the basis of the ATS implementation of the Levenberg-Marquardt algorithm.

6. Appendix B. Gradient Test for Correctness

One helpful check is to compute the gradient of the objective function at the computed minimum. The correct least-squares solution yields $\nabla J = 0$. Also, these derivatives are important because several minimization algorithms require that these derivatives be included. The definitions of f and g are kept the same as they were in Sec. 3. For every nonlinear geometry, assume the same definition for the objective function, J, as specified in Sec. 3. For the linear geometries, the objective functions are here defined in terms of unconstrained parameters.

Planes

$$J(\boldsymbol{x}, \boldsymbol{A}) = \sum g_i^2$$

$$\frac{\partial J}{\partial x} = 2\sum g_i (g_i)_x$$
$$\frac{\partial J}{\partial y} = 2\sum g_i (g_i)_y$$
$$\frac{\partial J}{\partial z} = 2\sum g_i (g_i)_z$$
$$\frac{\partial J}{\partial A} = 2\sum g_i (g_i)_A$$
$$\frac{\partial J}{\partial B} = 2\sum g_i (g_i)_B$$
$$\frac{\partial J}{\partial C} = 2\sum g_i (g_i)_C$$

Lines

$$J(\boldsymbol{x}, \boldsymbol{A}) = \sum f_i^2$$

$$\frac{\partial J}{\partial x} = 2\sum f_i (f_i)_x$$
$$\frac{\partial J}{\partial y} = 2\sum f_i (f_i)_y$$
$$\frac{\partial J}{\partial z} = 2\sum f_i (f_i)_z$$
$$\frac{\partial J}{\partial A} = 2\sum f_i (f_i)_A$$
$$\frac{\partial J}{\partial B} = 2\sum f_i (f_i)_B$$
$$\frac{\partial J}{\partial C} = 2\sum f_i (f_i)_C$$

Spheres

$$\frac{\partial J}{\partial x} = -2\sum d_i (x_i - x)/|\boldsymbol{x}_i - \boldsymbol{x}|$$
$$\frac{\partial J}{\partial y} = -2\sum d_i (y_i - y)/|\boldsymbol{x}_i - \boldsymbol{x}|$$
$$\frac{\partial J}{\partial z} = -2\sum d_i (z_i - z)/|\boldsymbol{x}_i - \boldsymbol{x}|$$
$$\frac{\partial J}{\partial r} = -2\sum d_i$$

2D Circles

$$\frac{\partial J}{\partial x} = -2\sum d_i (x_i - x)/(d_i + r)$$
$$\frac{\partial J}{\partial y} = -2\sum d_i (y_i - y)/(d_i + r)$$
$$\frac{\partial J}{\partial r} = -2\sum d_i$$

3D Circles

$$\frac{\partial J}{\partial x} = 2\sum [g_i (g_i)_x + (f_i - r)(f_i)_x]$$
$$\frac{\partial J}{\partial y} = 2\sum [g_i (g_i)_y + (f_i - r)(f_i)_y]$$
$$\frac{\partial J}{\partial z} = 2\sum [g_i (g_i)_z + (f_i - r)(f_i)_z]$$
$$\frac{\partial J}{\partial A} = 2\sum [g_i (g_i)_A + (f_i - r)(f_i)_A]$$
$$\frac{\partial J}{\partial B} = 2\sum [g_i (g_i)_B + (f_i - r)(f_i)_B]$$
$$\frac{\partial J}{\partial C} = 2\sum [g_i (g_i)_C + (f_i - r)(f_i)_C]$$
$$\frac{\partial J}{\partial r} = -2\sum (f_i - r)$$

Cylinders

$$\frac{\partial J}{\partial x} = 2\sum (f_i - r)(f_i)_x$$
$$\frac{\partial J}{\partial y} = 2\sum (f_i - r)(f_i)_y$$
$$\frac{\partial J}{\partial z} = 2\sum (f_i - r)(f_i)_z$$

$$\frac{\partial J}{\partial A} = 2\sum (f_i - r)(f_i)_A$$
$$\frac{\partial J}{\partial B} = 2\sum (f_i - r)(f_i)_B$$
$$\frac{\partial J}{\partial C} = 2\sum (f_i - r)(f_i)_C$$
$$\frac{\partial J}{\partial r} = -\,2\sum (f_i - r)$$

Cones

$$\frac{\partial J}{\partial x} = 2\sum d_i[(f_i)_x \cos\psi + (g_i)_x \sin\psi]$$
$$\frac{\partial J}{\partial y} = 2\sum d_i[(f_i)_y \cos\psi + (g_i)_y \sin\psi]$$
$$\frac{\partial J}{\partial z} = 2\sum d_i[(f_i)_z \cos\psi + (g_i)_z \sin\psi]$$
$$\frac{\partial J}{\partial A} = 2\sum d_i[(f_i)_A \cos\psi + (g_i)_A \sin\psi]$$
$$\frac{\partial J}{\partial B} = 2\sum d_i[(f_i)_B \cos\psi + (g_i)_B \sin\psi]$$
$$\frac{\partial J}{\partial C} = 2\sum d_i[(f_i)_C \cos\psi + (g_i)_C \sin\psi]$$
$$\frac{\partial J}{\partial \psi} = 2\sum d_i[-\,f_i \sin\psi + g_i \cos\psi]$$
$$\frac{\partial J}{\partial s} = -\,2\sum d_i$$

Tori

$$\frac{\partial J}{\partial x} = 2\sum \frac{d_i}{d_i + R}[g_i(g_i)_x + (f_i - r)(f_i)_x]$$
$$\frac{\partial J}{\partial y} = 2\sum \frac{d_i}{d_i + R}[g_i(g_i)_y + (f_i - r)(f_i)_y]$$
$$\frac{\partial J}{\partial z} = 2\sum \frac{d_i}{d_i + R}[g_i(g_i)_z + (f_i - r)(f_i)_z]$$
$$\frac{\partial J}{\partial A} = 2\sum \frac{d_i}{d_i + R}[g_i(g_i)_A + (f_i - r)(f_i)_A]$$
$$\frac{\partial J}{\partial B} = 2\sum \frac{d_i}{d_i + R}[g_i(g_i)_B + (f_i - r)(f_i)_B]$$
$$\frac{\partial J}{\partial C} = 2\sum \frac{d_i}{d_i + R}[g_i(g_i)_C + (f_i - r)(f_i)_C]$$
$$\frac{\partial J}{\partial r} = 2\sum \frac{d_i}{d_i + R}[g_i(g_i)_r - (f_i - r)]$$
$$\frac{\partial J}{\partial R} = -\,2\sum d_i$$

7. References

[1] T. H. Hopp, Computational Metrology—Vol. 6, No. 4, Manufacturing Review, American Society of Mechanical Engineers, New York (1993) pp. 295–304.

[2] R. Walker, CMM Form Tolerance Algorithm Testing, GIDEP Alert X1-A-88-01, Government-Industry Data Exchange Program, Department of Defense, Washington, D.C. (1988).

[3] D. A. Rosenfeld, User's Guide for the Algorithm Testing System Version 2.0, NISTIR 5674, National Institute of Standards and Technology, Gaithersburg, MD (1995).

[4] ASME, Methods for Performance Evaluation of Coordinate Measuring System Software, B89.4.10-199x, DRAFT, American Society of Mechanical Engineers, New York (1998).

[5] C. Diaz and T. H. Hopp, NIST Special Test Service: The Algorithm Testing and Evaluation Program for Coordinate Measuring Systems, NISTSP 250-41, National Institute of Standards and Technology, Gaithersburg, MD (to be published).

[6] C. Diaz and T. H. Hopp, Testing Coordinate Measuring Systems Software, Proceedings of ASQC Measurement Quality Conference, October 26–27, National Institutes of Standards and Technology, Gaithersburg, MD (1993).

[7] M. E. A. Algeo and T. H. Hopp, Form Error Models of the NIST Algorithm Testing System, NISTIR 4740, National Institute of Standards and Technology, Gaithersburg, MD, January 1992.

[8] T. M. Apostol, Calculus, Volume II, Second Edition, John Wiley & Sons, Inc., New York (1969) pp. 314–318.

[9] G. H. Golub and C. F. van Loan, Matrix Computations, The Johns Hopkins University Press, Baltimore, MD (1983).

[10] G. W. Stewart, Matrix Computations, Academic Press, Inc., Orlando, FL (1973) p. 320.

[11] J. C. Nash, Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation, Adam Hilger, Ltd., Bristol, England (1979).

[12] J. J. Moré, The Levenberg-Marquardt Algorithm: Implementation and Theory, in Numerical Analysis: Proc. Biennial Conf. on Numerical Analysis, Dold and Eckmann, eds., Dundee, Scotland (1977).

About the author: *Craig Shakarji is a mathematician in the Manufacturing Systems Integration Division of the NIST Manufacturing Engineering Laboratory. The National Institute of Standards and Technology is an agency of the Technology Administration, U.S. Department of Commerce.*