A Comparison of Four Algorithms for Estimating 3-D Rigid Transformations[1]

A. Lorusso

Robotics and Automation Department
Tecnopolis CSATA Novus Ortus
70010 Valenzano - Bari - Italy

adele@rob.csata.it

D. W. Eggert and R. B. Fisher

Department of Artificial Intelligence
University of Edinburgh
Edinburgh, Scotland EH1 2QL

eggertd or rbf@aifh.ed.ac.uk

Abstract

A common need in machine vision is to compute the 3-D rigid transformation that exists between two sets of points in a coordinate system for which corresponding pairs have been determined. Several solutions to this problem have been devised. In this paper a comparative analysis of four popular algorithms is given. Each of them computes the translation and rotation of the transform in closed-form, as the solution to a least squares formulation of the problem. They differ in terms of the representation of the transform and the method of solution, using respectively: singular value decomposition of a matrix, orthonormal matrices, unit quaternions and dual quaternions. This comparison presents results of several experiments designed to determine (1) the accuracy of each algorithm in the presence of different levels of noise, (2) the stability of each technique with respect to degenerate data sets, and (3) the relative computation time of each approach for different sizes and forms of data.

Keywords: Motion analysis, 3-D rigid transformations, pose estimation

Address for correspondence: David W. Eggert
Department of Artificial Intelligence
University of Edinburgh
5 Forrest Hill
Edinburgh, Scotland EH1 2QL

[1]This work was funded by EC H.C.M. Project ERB40500PL921003 through the SMART network, and UK EPSRC Grant GR/H/86905.

A Comparison of Four Algorithms for Estimating 3-D Rigid Transformations

Abstract

A common need in machine vision is to compute the 3-D rigid transformation that exists between two sets of points in a coordinate system for which corresponding pairs have been determined. Several solutions to this problem have been devised. In this paper a comparative analysis of four popular algorithms is given. Each of them computes the translation and rotation of the transform in closed-form, as the solution to a least squares formulation of the problem. They differ in terms of the representation of the transform and the method of solution, using respectively: singular value decomposition of a matrix, orthonormal matrices, unit quaternions and dual quaternions. This comparison presents results of several experiments designed to determine (1) the accuracy of each algorithm in the presence of different levels of noise, (2) the stability of each technique with respect to degenerate data sets, and (3) the relative computation time of each approach for different sizes and forms of data.

Summary

1. *What is the original contribution of this work?*

   This work presents a comparative analysis of four popular closed-form methods of computing a 3-D rigid transformation between two overdetermined sets of corresponded data: using singular value decomposition [1], orthonormal matrices [3], unit quaternions [2] or dual quaternions [8]. Experiments determining the accuracy of the algorithms in the presence of noise, their stability in the presence of degenerate data, and relative computation time are reported.

2. *Why should this contribution be considered important?*

   It is important to know under which conditions it is appropriate to use a particular algorithm. The comparison given here details where each of the four algorithms is likely to break down with respect to noise and degenerate data sets. Relative efficiency in terms of computation time is also an important factor in determining which algorithm to use. The work presented here can be used as a guide in weighing the trade-offs of speed and accuracy for selecting the appropriate algorithm.

3. *What is the most closely related work by others and how does this work differ?*

   A recent review by Sabata and Aggarwal [6] compares the theoretical aspects of three of the approaches reviewed here (SVD, orthonormal matrices and unit quaternions), but does not provide any quantitative comparison. In each of the four papers [1, 2, 3, 8] themselves, qualitative assessments of accuracy and stability are mentioned. In addition, a comparison of computation time between the SVD and a unit quaternion method is provided in [1], while a comparison of accuracy and computation time in the presence of noise between the SVD and dual quaternion approaches is presented in [8]. Thus, to the best of our knowledge, this work is the first to provide a quantitative and unbiased comparison of all four algorithms with respect to accuracy, stability and computation time.

4. *How can other researchers make use of the results of this work?*

   Since the four algorithms compared here are commonly used, it is hoped that this analysis will be a valuable aid to those trying to choose which technique is best for a particular application. This will save people time independently trying to generate such a comparison, and hopefully keep them from making a wrong decision.

1 Introduction

Finding the relationship between two coordinate systems by using sets of corresponded feature measurements in each system is known as the *absolute orientation problem.* It has numerous applications in the areas of *photogrammetry*, robotics (*constructing world models*), *object motion analysis*, relating camera coordinate systems to others (*the hand-eye transform*), as well as estimating the position and orientation of recognized objects (*pose estimation*).

A recent survey by Sabata and Aggarwal [6] lists the large number of methods that have been developed to compute the 3-D rigid transformation between two sets of corresponded features. However, it does not provide any quantitative comparisons of these techniques. In this paper a comparison is performed on a subset of these approaches: those which use *corresponded points* as the feature sets. The problem of determining point correspondence, itself an active research area, is not addressed.

The algorithms for computing the transformation between two point sets fall into two basic categories: the earlier *iterative* methods and the more recent techniques which compute a *closed-form* solution. Iterative processes typically suffer from problems in guaranteeing convergence, avoiding local minima of the error function and computing a good starting estimate. This comparison restricts itself to the latter approaches, which compute the best transformation in a single step.

Four popular closed-form solutions are compared here. These differ according to the representations used for the transformation components, and alternative ways of minimizing the criterion function. The first was developed by Arun, Huang and Blostein [1], and is based on computing the *singular value decomposition* (SVD) of a derived matrix. A similar approach based on *orthonormal matrices* and computing an eigensystem of a derived matrix was presented by Horn, Hilden and Negahdaripour [3]. The third algorithm was also developed by Horn [2], and involves representing the rotation component of the transformation using a *unit quaternion.* The use of *dual quaternions* to represent both the translation and rotation components is the basis of the fourth technique as presented by Walker, Shao and Volz [8].

The comparison presented here consists of three parts. First, the *accuracy* of each algorithm is examined as the coordinates of corresponding points are corrupted with increasing amounts of noise. Second, the *stability* of the algorithms is determined as the original 3-D point sets degenerate into such forms as a plane, line and single point. Lastly, the relative *efficiency* of the algorithms, in terms of actual execution time, is reported for the above situations. It is hoped that these results will make the choice of an appropriate algorithm for a given application simpler and more reliable.

In the next section, the basics of each of the four algorithms are presented to provide a framework for the comparison. The series of experiments, and their respective results, are detailed in section three. Conclusions about the overall performance of the techniques are given in section four.

2 Description of Algorithms

Each of the four algorithms computes the solution to a similar problem which can be described as follows. Assume that there exist two corresponded point sets $\{m_i\}$ and $\{d_i\}$, $i = 1 \ldots N$, such that they are related by:

$$d_i \;=\; R\, m_i + T + V_i \tag{1}$$

where R is a standard 3×3 rotation matrix, T is a 3-D translation vector and V_i a noise vector. The problem is to solve for the optimal transformation $[\ \hat{R}\ ,\ \hat{T}\ ]$ that maps the set $\{m_i\}$ onto $\{d_i\}$. This solution is that which minimizes the *least squares error criterion* given by:

$$\Sigma^2 \;=\; \sum_{i=1}^{N} \parallel d_i - \hat{R}\, m_i - \hat{T} \parallel^2 \tag{2}$$

Just how this equation is minimized is the subject of the next four subsections. For each method the basic steps involved in minimizing equation (2) are listed. The complete derivations are not included here, but the reader is invited to consult the appropriate paper for more details. Included, however, is exactly how the rotation and translation components of the transformation are represented, as well as any special cases that must be considered for a complete solution.

2.1 A solution involving the singular value decomposition of a matrix

This first method was developed by Arun, Huang and Blostein [1], and was designed to explicitly minimize equation (2).

2.1.1 Transformation representation

For this approach, the rotation is represented using a standard 3×3 orthonormal matrix. Translation is a 3-D vector, as in equation (1).

2.1.2 Calculating rotation

As a consequence of a least-squares solution to equation (2), the point sets $\{d_i\}$ and the transformed $\{m_i\}$ have the same centroid. Using this constraint a new equation is generated. By defining:

$$
\begin{aligned}
\bar{d} &= \frac{1}{N} \sum_{i=1}^{N} d_i & \qquad \bar{m} &= \frac{1}{N} \sum_{i=1}^{N} m_i \\
d_{c,i} &= d_i - \bar{d} & \qquad m_{c,i} &= m_i - \bar{m}
\end{aligned}
$$

equation (2) becomes:

$$
\Sigma^2 = \sum_{i=1}^{N} \| \, d_{c,i} - \hat{R} \, m_{c,i} \, \|^2 = \sum_{i=1}^{N} (d_{c,i}^T \, d_{c,i} + m_{c,i}^T \, m_{c,i} - 2 \, d_{c,i}^T \, \hat{R} \, m_{c,i}) \tag{3}
$$

This equation is minimized when the last term is maximized, which is equivalent to maximizing the $Trace(\hat{R}H)$, where

$$
H = \sum_{i=1}^{N} m_{c,i} \, d_{c,i}^T
$$

If the SVD of H is given by $H = U \Lambda V^T$, then the optimal rotation matrix, $\hat{R}$, that maximizes the trace is

$$
\hat{R} = V \, U^T \tag{4}
$$

Note: Equation (3) is also known as the *orthogonal Procrustes problem*, the SVD-based solution of which has been known for some time [7].

2.1.3 Calculating translation

The optimal translation aligns the centroid of the set $\{d_i\}$ with the centroid of the rotated set $\{m_i\}$. That is

$$
\hat{T} = \bar{d} - \hat{R} \, \bar{m} \tag{5}
$$

2.1.4 Special cases

When the two point sets are planar, if the determinant of $\hat{R}$ is positive one, all is well. If the determinant is negative one, then a solution representing a reflection rather than a rotation has been computed. In this case the desired rotation is found as $\hat{R} = V'U^T$, where $V' = [\, v_1, v_2, -v_3 \,]$ and v_3 is the third column of V corresponding to the singular value of H that is zero. The algorithm is not designed for linear or singular point data sets.

2.2 A solution involving orthonormal matrices

The second algorithm is similar in nature to the first, but was developed independently by Horn, Hilden and Negahdaripour [3]. As presented in their paper, the criterion error function was slightly different:

$$\Sigma^2 = \sum_{i=1}^{N} \| \, d_i - \hat{s}\, \hat{R}\, m_i - \hat{T} \, \|^2 \tag{6}$$

including a scale factor, $\hat{s}$, in the transformation. Solving for $\hat{s}$ is independent of solving for $\hat{R}$ and $\hat{T}$. In the following summary of their approach it is assumed that $\hat{s} = 1$ to enable an easier comparison.

2.2.1 Transformation representation

As in the SVD approach, rotation is represented using the 3×3 orthonormal matrix and translation is given by a 3-D vector.

2.2.2 Calculating rotation

As in the SVD algorithm, the constraint of coinciding point set centroids is used to derive equation (3). Again, this equation is minimized as the last term is maximized, being also equivalent to maximizing the $Trace(\hat{R}^T M)$, where

$$M = H^T = \sum_{i=1}^{N} d_{c,i}\, m_{c,i}^T$$

Assuming M is non-singular, it can be decomposed as $M = US$, where

$$U = M\, S^{-1} \qquad\qquad S = (M^T M)^{1/2}$$

U being an orthonormal matrix. In the case where the $Trace(\hat{R}^T M)$ is maximized, $\hat{R} = U$ and can be expressed as:

$$\hat{R} \;=\; M\,(\frac{1}{\sqrt{\lambda_1}}\,u_1\,u_1^T + \frac{1}{\sqrt{\lambda_2}}\,u_2\,u_2^T + \frac{1}{\sqrt{\lambda_3}}\,u_3\,u_3^T) \tag{7}$$

where $\{\lambda_i\}$ and $\{u_i\}$ are the eigenvalues and corresponding eigenvectors of the matrix $M^T M$.

2.2.3 Calculating translation

The optimal translation $\hat{T}$ is computed using equation (5) given $\hat{R}$ as found in (7).

2.2.4 Special cases

In the case where the point sets are planar, the matrix M is singular and is of rank two. The solution above does not hold in this case. The alternative solution presented in [3] is:

$$\hat{R} \;=\; M\,(\frac{1}{\sqrt{\lambda_1}}\,u_1\,u_1^T + \frac{1}{\sqrt{\lambda_2}}\,u_2\,u_2^T) \pm u_3\,u_3^T \tag{8}$$

where u_3 is the eigenvector corresponding to the zero eigenvalue of the matrix $M^T M$. The plus or minus sign is chosen such that the determinant of $\hat{R}$ is positive one.

Unfortunately, this solution is not correct in all cases and must be modified. One such modification is given by:

$$\hat{R} \;=\; M\,S^+ \pm \frac{X}{\sqrt{|\,Trace(X)\,|}} \tag{9}$$

where

$$S^+ \;=\; (\frac{1}{\sqrt{\lambda_1}}\,u_1\,u_1^T + \frac{1}{\sqrt{\lambda_2}}\,u_2\,u_2^T) \qquad\qquad X \;=\; [\,(M\,S^+)\,(M\,S^+)^T - I\,]\,u_3\,u_3^T$$

This formulation will also hold in the general case of non-planar point sets, but is more expensive to compute. However, if the point set is linear or singular, the eigenvalues λ_2 and λ_1 become zero which invalidates the solution.

2.3 A solution involving unit quaternions

The third method is also due to Horn [2], and was originally designed to solve equation (6). However, in this case a different technique was used based on a different representation of the transformation. Again, assume $\hat{s} = 1$ in the following.

2.3.1 Transformation representation

Rather than use the standard 3×3 orthonormal matrix to represent rotation, the unit quaternion is employed. If the rotation is represented by a movement through an angle θ about an axis $a = [\ a_x, a_y, a_z\ ]$ through the origin, the equivalent unit quaternion is a 4-D vector defined as $q = [\ cos(\theta/2),\ sin(\theta/2)\ a_x,\ sin(\theta/2)\ a_y,\ sin(\theta/2)\ a_z\ ]$. Translation is still represented using a 3-D vector in the following algorithm.

2.3.2 Calculating rotation

Unit quaternions have several useful properties (see [2]) which can be used to rewrite the final term of equation (3) in terms of quaternions and quaternion multiplication as:

$$E \;=\; \sum_{i=1}^{N} d_{c,i}^{T}\ \hat{R}\ m_{c,i} \;=\; \sum_{i=1}^{N} (\hat{q}\ \dot{m}_{c,i}\ \hat{q}^{\,*}) \cdot \dot{d}_{c,i} \tag{10}$$

where $\hat{q}$ is the unit quaternion representing $\hat{R}$, $\hat{q}^{\,*} = [\ \hat{q}_0, -\hat{q}_1, -\hat{q}_2, -\hat{q}_3\ ]$ and $\dot{m}_{c,i}$ and $\dot{d}_{c,i}$ are quaternions formed by setting the first component to zero and the remaining three to the point's coordinates. The expression in the above equation can be further rewritten as $E = \hat{q}^{\,T} P\ \hat{q}$ where P is a 4×4 matrix given by:

$$P \;=\; \begin{bmatrix} (S_{xx}+S_{yy}+S_{zz}) & S_{yz}-S_{zy} & S_{zx}-S_{xz} & S_{xy}-S_{yx} \\ S_{yz}-S_{zy} & (S_{xx}-S_{yy}-S_{zz}) & S_{xy}+S_{yx} & S_{zx}+S_{xz} \\ S_{zx}-S_{xz} & S_{xy}+S_{yx} & (-S_{xx}+S_{yy}-S_{zz}) & S_{yz}+S_{zy} \\ S_{xy}-S_{yx} & S_{zx}+S_{xz} & S_{yz}+S_{zy} & (-S_{xx}-S_{yy}+S_{zz}) \end{bmatrix}$$

with the S terms being sums of products of point coordinates as in the elements of H earlier:

$$S_{ab} \;=\; \sum_{i=1}^{N} m_{c,i_a}\ d_{c,i_b}$$

The quaternion $\hat{q}$ which maximizes equation (10) is the eigenvector corresponding to the largest positive eigenvalue of matrix P. Given $\hat{q}$ the equivalent rotation matrix $\hat{R}$ can be found as:

$$\hat{R} \;=\; \begin{bmatrix} (\hat{q}_0^2+\hat{q}_1^2-\hat{q}_2^2-\hat{q}_3^2) & 2\,(\hat{q}_1\,\hat{q}_2-\hat{q}_0\,\hat{q}_3) & 2\,(\hat{q}_1\,\hat{q}_3+\hat{q}_0\,\hat{q}_2) \\ 2\,(\hat{q}_2\,\hat{q}_1+\hat{q}_0\,\hat{q}_3) & (\hat{q}_0^2-\hat{q}_1^2+\hat{q}_2^2-\hat{q}_3^2) & 2\,(\hat{q}_2\,\hat{q}_3-\hat{q}_0\,\hat{q}_1) \\ 2\,(\hat{q}_3\,\hat{q}_1-\hat{q}_0\,\hat{q}_2) & 2\,(\hat{q}_3\,\hat{q}_2+\hat{q}_0\,\hat{q}_1) & (\hat{q}_0^2-\hat{q}_1^2-\hat{q}_2^2+\hat{q}_3^2) \end{bmatrix} \tag{11}$$

2.3.3 Calculating translation

Once the matrix $\hat{R}$ has been computed from the elements of $\hat{q}$, the optimal translation $\hat{T}$ can again be found using equation (5).

2.3.4 Special cases

If the point sets are planar the algorithm does not have to modified, although depending on the method used to compute the eigensystem of matrix P certain simplifications can be made. Horn does present another alternate algorithm for planar point sets in his paper, but it will not be used in this comparison. None of these methods are intended for linear or singular point data sets.

2.4 A solution involving dual quaternions

The fourth algorithm is due to Walker, Shao and Volz [8], and is the most significantly different of the set. The original error criterion function which they minimized is:

$$\Sigma^2 = \sum_{i=1}^{L} \alpha_i \parallel n_{1i} - \hat{R}\, n_{2i} \parallel^2 + \sum_{i=1}^{N} \beta_i \parallel d_i - \hat{R}\, m_i - \hat{T} \parallel^2 \qquad (12)$$

where $\{n_{1i}\}$ and $\{n_{2i}\}$ are two sets of L corresponding unit normal vectors, and $\{\alpha_i\}$, $\{\beta_i\}$ are weighting factors reflecting data reliability. Thus this approach incorporates both positional and directional information in the minimization. In order to compare this approach with the others, it is assumed that $\alpha_i = 0$ and $\beta_i = 1$ so that only point positions are relevant. Note that if independent direction information is available, however, this algorithm can integrate both data types, unlike the previous algorithms in their current form.

2.4.1 Transformation representation

In this method the rotation and translation are both represented using a dual quaternion. As the name suggests, it is a quaternion consisting of two parts, $q_d = [r, s]$. Rather than describing motion as a rotation about the origin and then a translation, it can be modeled as a simultaneous rotation and translation along a particular line in 3-D space. If this line has direction $n = [\ n_x, n_y, n_z\ ]$, passing through a point $p = [\ p_x, p_y, p_z\ ]$, and the amount of motion is given by an angle θ and a distance t, then the two components of q_d are 4-D vectors defined as:

$$r = \begin{bmatrix} \sin(\theta/2)\ n \\ \cos(\theta/2) \end{bmatrix} \qquad\qquad s = \begin{bmatrix} \frac{t}{2}\ \cos(\theta/2)\ n + \sin(\theta/2)\ (p \times n) \\ -\frac{t}{2}\ \sin(\theta/2) \end{bmatrix}$$

The two components of the dual quaternion have the properties that $r\, r^T = 1$ and $s^T r = 0$.

2.4.2 Calculating rotation

The restricted version of equation (12) can be rewritten in terms of the components of the dual quaternion $\hat{q}_d = [\hat{r}, \hat{s}]$ as:

$$\begin{aligned}
\Sigma^2 &= \sum_{i=1}^{N} (\hat{s}^T \hat{s} + 2\, \hat{s}^T (W(m_i^{'}) - Q(d_i^{'}))\, \hat{r} - 2\, \hat{r}^T Q(d_i^{'})\, W(m_i^{'})\, \hat{r} + (m_i^{'T} m_i^{'} + d_i^{'T} d_i^{'}) \\
&= \hat{r}^T C_1\, \hat{r} + \hat{s}^T C_2\, \hat{s} + \hat{s}^T C_3\, \hat{r} + C_4 \qquad (13)
\end{aligned}$$

where $m_i^{'}$ and $d_i^{'}$ are 4-D vectors with the first three components set equal to half the values of the original point's coordinates and the fourth component equal to zero. The 4×4 matrices $Q(v)$ and $W(v)$ are defined by:

$$Q(v) = \begin{bmatrix} v_3\, I + K(v_{0..2}) & v_{0..2} \\ -v_{0..2}^T & v_3 \end{bmatrix} \qquad W(v) = \begin{bmatrix} v_3\, I - K(v_{0..2}) & v_{0..2} \\ -v_{0..2}^T & v_3 \end{bmatrix}$$

$$K(v) = \begin{bmatrix} 0 & -v_2 & v_1 \\ v_2 & 0 & -v_0 \\ -v_1 & v_0 & 0 \end{bmatrix}$$

where $v_{0..2}$ represents the first three elements of v. The C coefficients are given as:

$$\begin{aligned}
C_1 &= \sum_{i=1}^{N} Q(d_i^{'})\, W(m_i^{'}) & C_2 &= N\, I \\
C_3 &= 2 \sum_{i=1}^{N} (W(m_i^{'}) - Q(d_i^{'})) & C_4 &= \sum_{i=1}^{N} (m_i^{'T} m_i^{'} + d_i^{'T} d_i^{'})
\end{aligned}$$

In order to properly minimize equation (13) the two constraints $r\, r^T = 1$ and $s^T r = 0$ are added using Lagrange multipliers. Then it can be derived that given the 4×4 matrix A:

$$A = \frac{1}{2} (C_3^T (C_2 + C_2^T)^{-1} C_3 - C_1 - C_1^T) = \frac{1}{4N} C_3^T C_3 - C_1$$

the optimal value of $\hat{r}$ is the eigenvector corresponding to the largest positive eigenvalue of A. The component $\hat{r}$ can be used to calculate the equivalent rotation matrix $\hat{R}$ as:

$$\hat{R} = (\hat{r}_3^2 - \hat{r}_{0..2}^T\, \hat{r}_{0..2})\, I + 2\, \hat{r}_{0..2}\, \hat{r}_{0..2}^T + 2\, r_3\, K(r_{0..2}) \qquad (14)$$

2.4.3 Calculating translation

The optimal value of $\hat{s}$ is calculated based on $\hat{r}$ as $\hat{s} = -(C_2 + C_2^T)^{-1} C_3\, \hat{r} = -\frac{1}{2N} C_3\, \hat{r}$. Given $\hat{s}$ the equivalent translation $\hat{T}$ is

$$\hat{T} = W(\hat{r})^T\, \hat{s} \qquad (15)$$

2.4.4 Special cases

As with the unit quaternion algorithm, there is no need for modification in the case where the points sets are planar, and linear or singular point sets are not handled.

3 Experimental Comparison

Each of the four algorithms was implemented and a series of experiments performed to determine their accuracy, stability and speed. In the following these experimental results are detailed.

3.1 The implementations

The implementations were coded in C++ and compiled using the GNU compiler[2]. While the majority of the coding was straightforward, a decision needed to be made concerning the algorithms used for computing the SVD and eigensystems of a matrix. In Horn's work [2, 3] it is stated that the eigensystems can be computed by analytically solving the cubic and quartic characteristic equations and then using Gaussian elimination (the SVD of a matrix could also be calculated analytically using this eigensystem algorithm). The other authors used routines from various mathematical packages. It was found that these standard routines were more stable than Horn's analytic approach. Therefore, to provide another source of commonality, routines from the Numerical Recipes package [5] were used in all. Singular value decomposition was performed using the *svdcmp* routine (which in turn is based on the LINPACK SVD algorithm), while eigensystems were computed using the *jacobi* algorithm. Since these algorithms are in fact iterative, the overall closed-form nature of the solution is technically violated. However, these routines are well known for their stability and rapid convergence (the Jacobi algorithm is actually guaranteed to converge), and no problems were observed during the experiments.

3.2 Accuracy experiment

In this experiment, the absolute accuracy of each of the algorithms was tested under varying levels of noise. The data for these tests was generated as follows. First, several different data set sizes of nondegenerately arranged 3-D points were used, ranging from a minimum of $N = 4$ points to a

[2]Code was compiled using gcc, version 2.5.8, using -O optimization

maximum of $N = 10,000$ points. The points in these data sets $\{m_i\}$ were chosen randomly from a uniform distribution within a cube of size $2 \times 2 \times 2$ centered about the origin. The corresponding data set $\{d_i\}$ was formed by adding noise to each point and transforming them to a new location.[3] The noise added to each component was Gaussian with zero mean and standard deviation ranging from a minimum of zero to maximum of one, and was uncorrelated between components.

Transformations were generated in two parts. A 3-D translation was computed by randomly selecting components uniformly distributed in the range $[-10 .. 10]$. A rotation was calculated from a unit quaternion whose components were randomly chosen in the range $[-1 .. 1]$ such that the magnitude of the quaternion was originally less than one, and then properly normalized. This technique generates rotations from a uniform distribution.

For each data set size and noise level one hundred trials were run, a trial consisting of a new set of points, amounts of noise and transformation. The average response of the algorithms over these one hundred trials was used to compute three different error statistics for the calculated transforms. These are the norm of the difference of the true and estimated translation vectors, as well as the norm of the difference of true and estimated unit quaternions representing the rotation. Also the root mean square (RMS) error of the distance between the points in the two data sets under the found transformation was calculated. These error measures were also computed between the results of the algorithms themselves to determine the levels of difference between them.

The left column of Figure 1 shows the accuracy of the four algorithms for the changes in noise level in terms of the error in translation, rotation and RMS error, respectively. As would be expected, as the number of points grows, the error in the computed transformation approaches a value dependent on the noise level used to disturb the data (the RMS error actually approaches $\sqrt{3}$ times the standard deviation of the noise, σ). The exception to this is the set of graphs for zero noise, which are expanded in the right column of Figure 1. Here it is seen that the dual quaternion algorithm is less accurate than the others, which are very similar except for small data sets.

To further examine the responses, Figure 2 shows the differences in the translation and rotation components between algorithms. Those shown are between the SVD and unit quaternion (2.a and

[3]Normally noise is added after the transformation. If the transform is Euclidean and the noise is isotropic, both techniques are equivalent. However, later experiments require special anisotropic noise to be added, which is most easily done before the transformation.

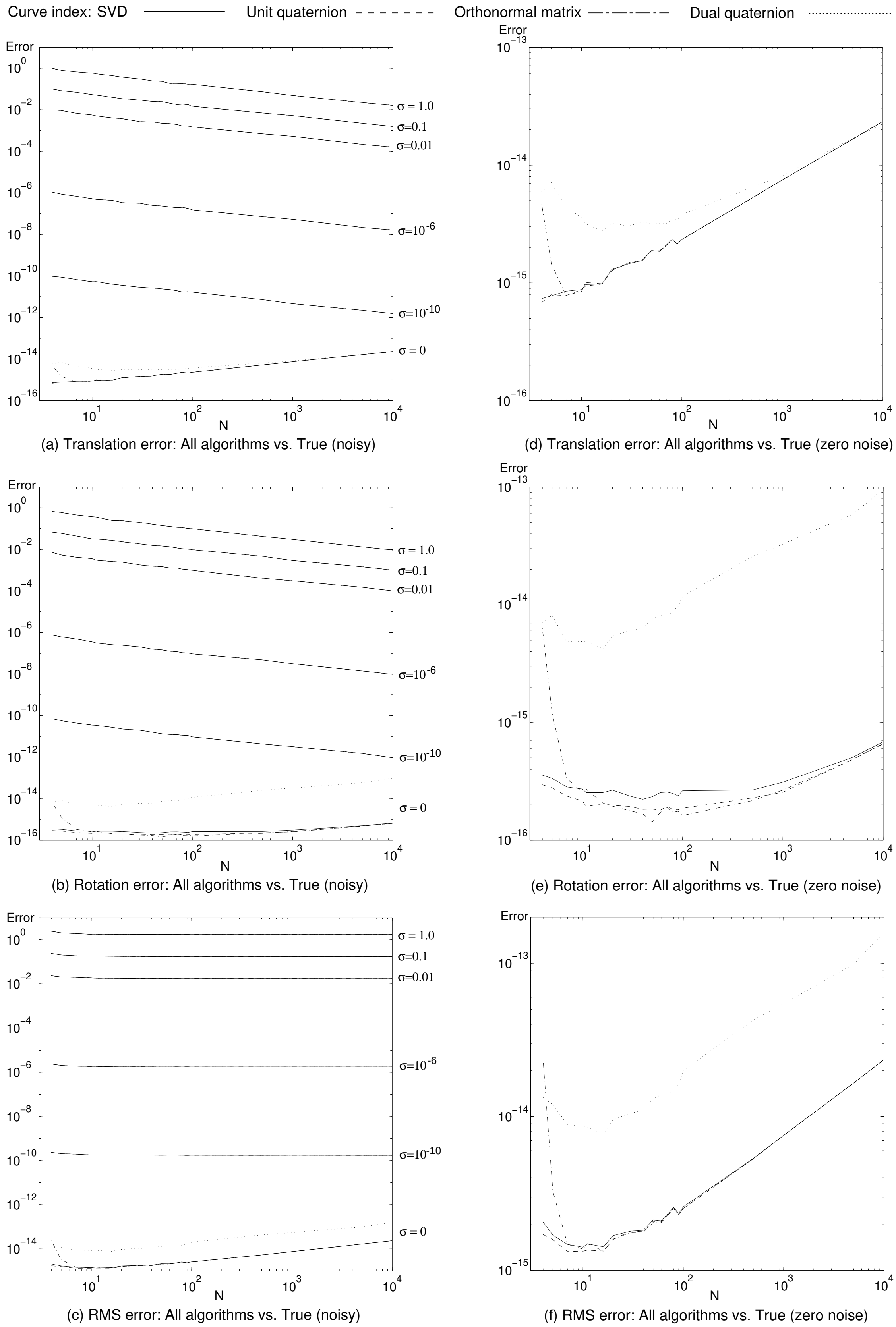
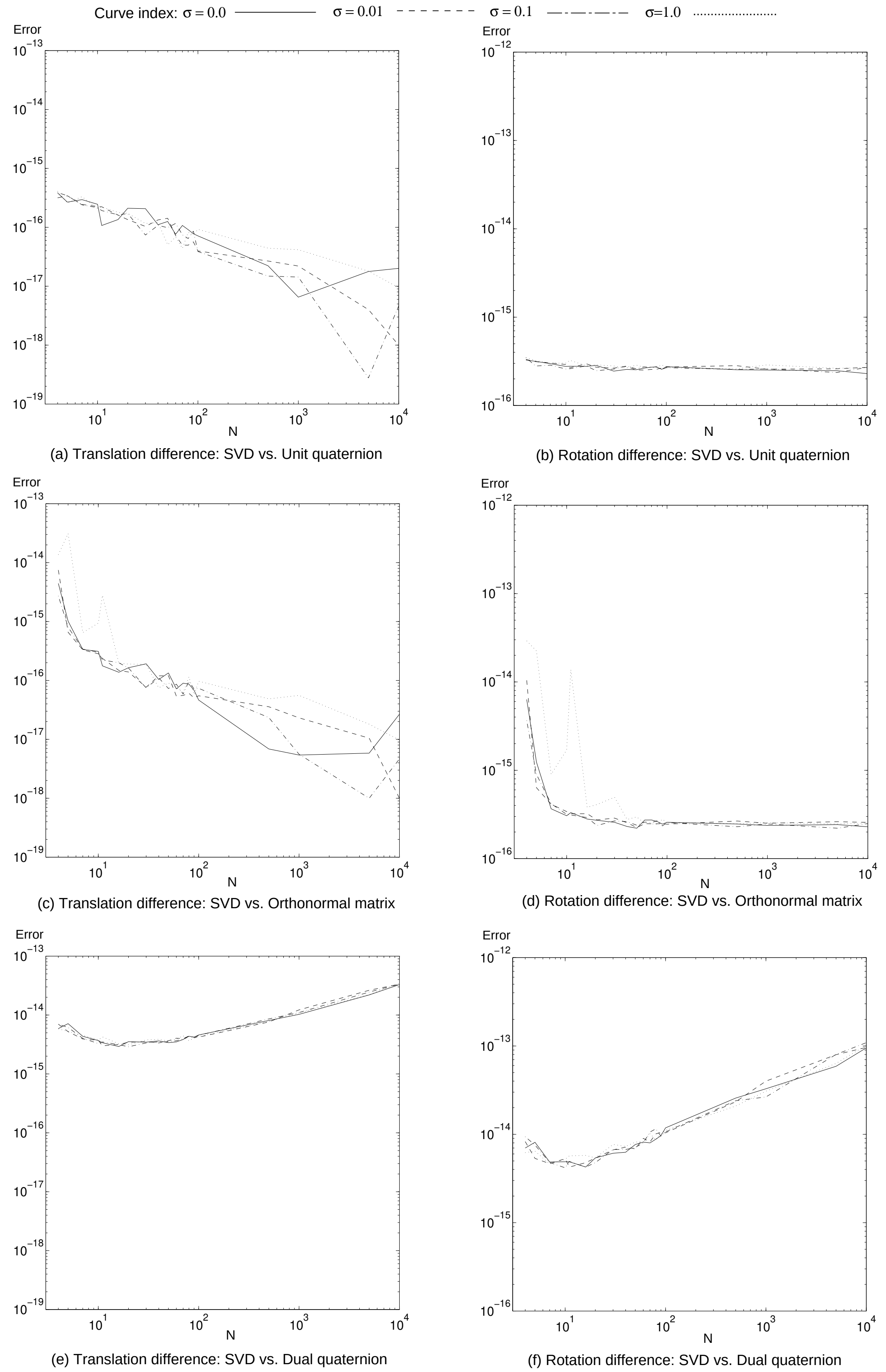


Figure 1: Errors between computed translation and rotation from algorithms and known transformation. Graphs (log-log scale) are translation error, $\|\hat{T}_{alg} - \hat{T}_{true}\|$, and rotation error, $\|\hat{q}_{alg} - \hat{q}_{true}\|$, vs. data set size, N. Noise levels $\sigma = 0.0, 10^{-10}, 10^{-6}, 0.01, 0.1, 1.0$ are shown in the left column (a) - (c), while the zero noise case is depicted in the right column (d) - (f).



(a) Translation difference: SVD vs. Unit quaternion

(b) Rotation difference: SVD vs. Unit quaternion

(c) Translation difference: SVD vs. Orthonormal matrix

(d) Rotation difference: SVD vs. Orthonormal matrix

(e) Translation difference: SVD vs. Dual quaternion

(f) Rotation difference: SVD vs. Dual quaternion

Figure 2: Differences between computed transformations of SVD and unit quaternion ((a) and (b)), orthonormal matrix ((c) and (d)) and dual quaternion ((e) and (f)) algorithms. Graphs (log-log scale) are translation error, $\|\hat{T}_{svd} - \hat{T}_{other}\|$, and rotation error, $\|\hat{q}_{svd} - \hat{q}_{other}\|$, vs. data set size, N, for noise levels $\sigma = 0.0$, 0.01, 0.1, 1.0.

2.b), orthonormal matrix (2.c and 2.d) and dual quaternion (2.e and 2.f) algorithms. As one can see, the differences are really minimal, and in most cases are near the level of machine precision ($2 * 10^{-16}$). Even the largest difference, 10^{-13}, between the dual quaternion and SVD algorithms (2.f), is not considered large. Also, given the magnitudes of the errors, the various shapes of the curves are not overly significant.

So, from the results shown here, one can conclude that the difference in accuracy between the algorithms is many orders of magnitude below the normal RMS noise levels, almost at the level of machine precision. The unit quaternion and SVD results are most similar, deviating from that of the orthonormal matrix only for small data sets, while these three perform better than the dual quaternion method.

3.3 Stability experiments

In this section the stability of the algorithms is tested, in terms of how well they compute a correct transformation under different data conditions. In most cases the algorithms were designed for nondegenerate 3-D data sets, with modifications for planar data, a degenerate 2-D data set. However, the algorithms were not designed to handle linear or singular point data sets due to the unconstrained degrees of freedom. The following subsections explore how well the algorithms perform as the data sets approach these degenerate forms.

The data sets for these experiments were generated in a manner similar to the last. However a sequence of data sets was needed in these cases. The first set in the sequence was identical to that chosen from the $2 \times 2 \times 2$ cube. Subsequent sets were taken from a volume whose dimensions were steadily reduced to approach those of a plane, a line, or a single point.

With this data sequence, two types of noise models were examined as well. The first is isotropic, as was used in the first experiment. This models the case where data is gathered and no prior uncertainty model is known. The second is anisotropic noise, which models a belief that the data is known to be fairly accurate in some dimension, but less so in others. The more accurate dimension in this case is the direction of degeneracy (for instance, perpendicular to a plane). The level of noise added in this direction was reduced. Adding this type of noise in some other direction only lead to responses as when isotropic noise was used.

Lastly, the error criterion used to measure the stability of the algorithms is the RMS error. However, it is not appropriate to compute the RMS error between the transformed data and the original model points. This is because several transforms can produce equivalent error values for degenerate data sets. For instance, rotations about the axis of a line of data would not move a data point far from its corresponding position as the diameter of this line becomes small. Thus a control data set is used, consisting of a number of points equal to that used in the computation, but taken from the original $2 \times 2 \times 2$ cube. This data is transformed and perturbed using the same level and type of noise as the original set, then the RMS calculation is done for these points using the estimated transformation. By using this control set, the effects of degenerating rotation estimates can be observed.

3.3.1 3-D to 2-D degeneracy

In this experiment the data set degenerates into a plane. The Z dimension of the 3-D cube is steadily shrunk from its 2 unit size down to the level of machine precision at $2 * 10^{-16}$. Figure 3 shows the error responses of each of the algorithms as the degeneracy ratio (ratio of original cube Z dimension to new reduced Z dimension) is increased, i.e. the data set more closely resembles a plane. Here the isotropic noise model is used. Curves are drawn for varying levels of noise and different size data sets. Figure 4 shows the similar responses when anisotropic noise is used. Here, the noise level, σ, is the same in the X and Y dimensions as for the isotropic case. The new value of σ in the Z dimension is divided by the degeneracy ratio, reducing it for more planar sets.

These graphs continue to support conclusions drawn from the first experiment. Namely, as the size of the data set increases the RMS error value converges (to $\sqrt{3}\ \sigma$ for isotropic noise and $\sqrt{2}\ \sigma$ in the anisotropic case). Also, the error values are for the most part independent of the degeneracy value, which was to be expected since the algorithms were supposed to handle planar point sets. The only deviations from this occur in the zero noise case. Here, the response from the orthonormal matrix is less numerically stable than the others (perhaps suggesting the correction provided in this paper is not the best alternative). Figure 5 provides a more direct comparison of the responses of the four algorithms. Figures 5.c - 5.f show that under noisy conditions the responses of the algorithms are the same, but when noise is not present (5.a - 5.b), only the SVD

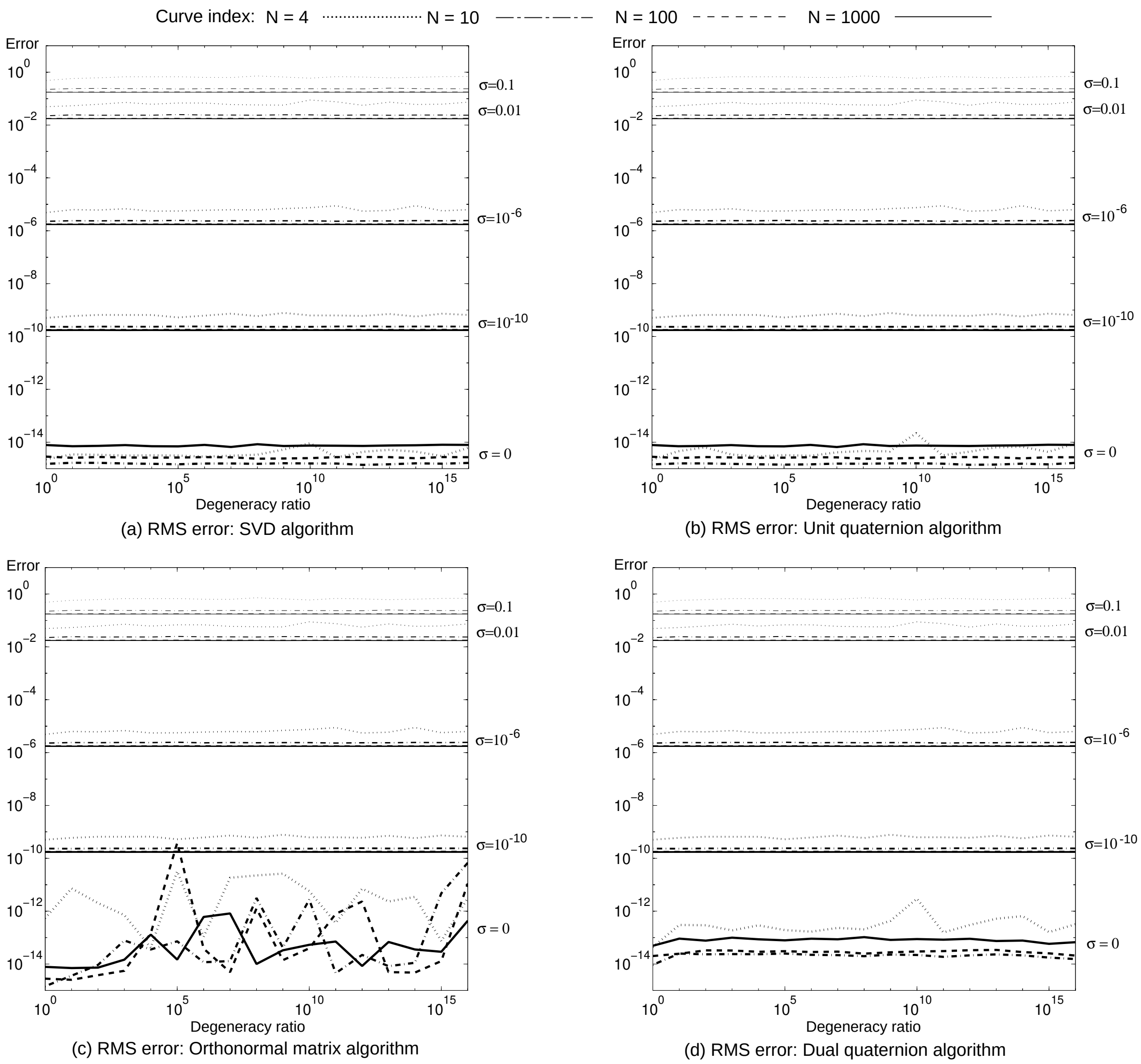


Figure 3: RMS errors of (a) SVD, (b) Unit quaternion, (c) Orthonormal matrix and (d) Dual quaternion algorithms for 2-D degenerate data sets under isotropic noise. Curves on graphs (log-log scale) present errors for four data set sizes, N = 4, 10, 100, 1000, and for five noise levels $\sigma = 0.0, 10^{-10}, 10^{-6}, 0.01, 0.1$. The thickest curves are $\sigma = 0$, while the thinnest are $\sigma = 0.1$.

and unit quaternion approaches remain similar, while both the dual quaternion and orthonormal matrix techniques produce less correct answers.

3.3.2 3-D to 1-D degeneracy

In this experiment the data set degenerates into a line. The X and Y dimensions of the 3-D cube are steadily shrunk from their 2 unit size down to the level of machine precision at $2 * 10-16$. Figures 6 and 7 show the error responses of the algorithms in the isotropic and anisotropic noise

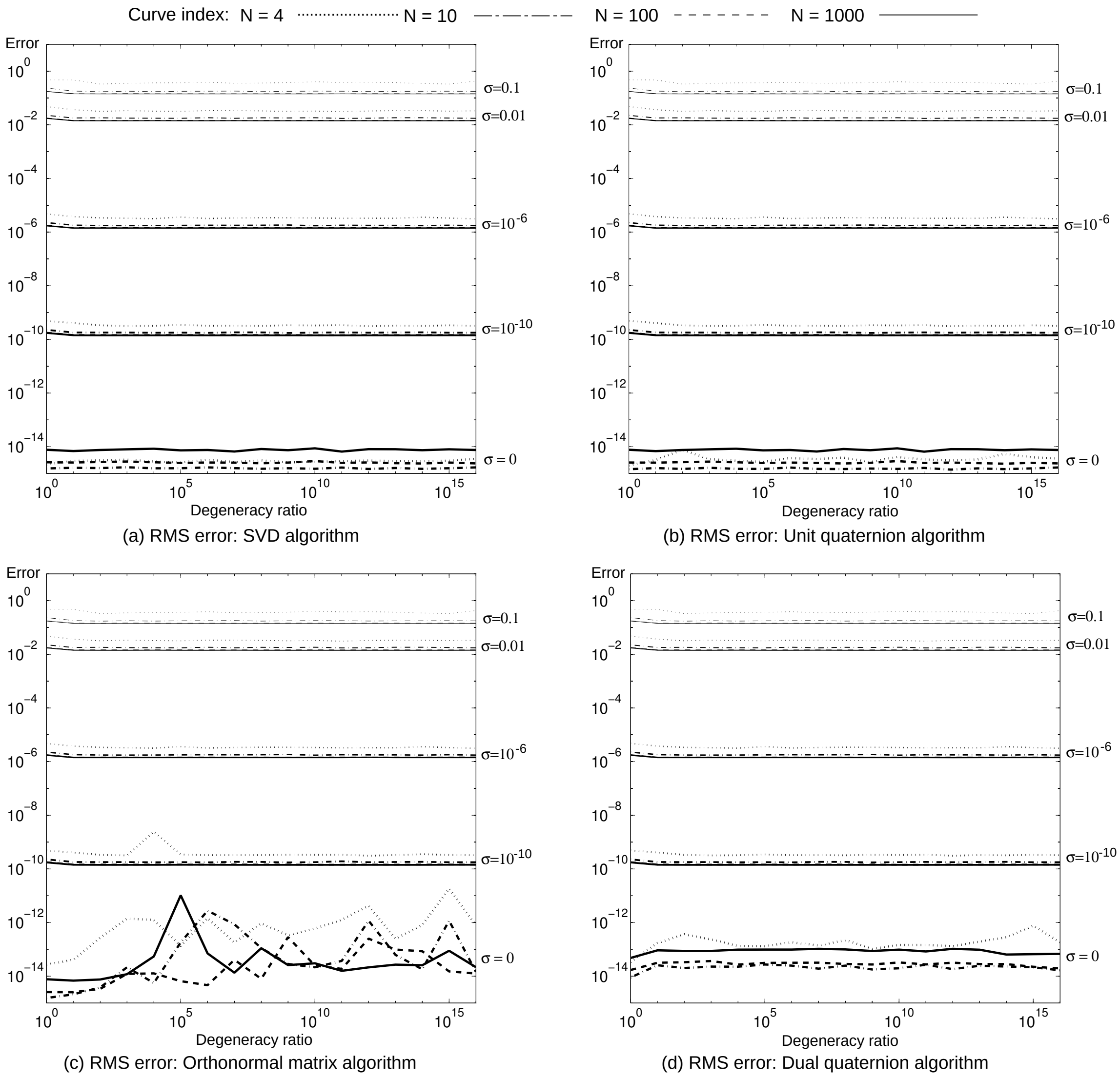


Figure 4: RMS errors of (a) SVD, (b) Unit quaternion, (c) Orthonormal matrix and (d) Dual quaternion algorithms for 2-D degenerate data sets under anisotropic noise. Curves on graphs (log-log scale) present errors for four data set sizes, N = 4, 10, 100, 1000, and for five noise levels $\sigma = 0.0, 10^{-10}, 10^{-6}, 0.01, 0.1$. The thickest curves are $\sigma = 0$, while the thinnest are $\sigma = 0.1$.

cases, again for five levels of noise and four data set sizes. In the anisotropic case the noise level in the Z dimension is the same as in the isotropic case, while that for the X and Y dimensions is reduced by the degeneracy ratio.

The algorithms are not designed to handle linear data sets. The graphs in Figures 6 and 7 show the different amounts of degeneracy necessary in the data to cause breakdown. For isotropic noise the error steadily rises as the diameter of the line decreases, completely breaking down in the best cases when the line diameter reaches the level of the noise, and in most other cases earlier

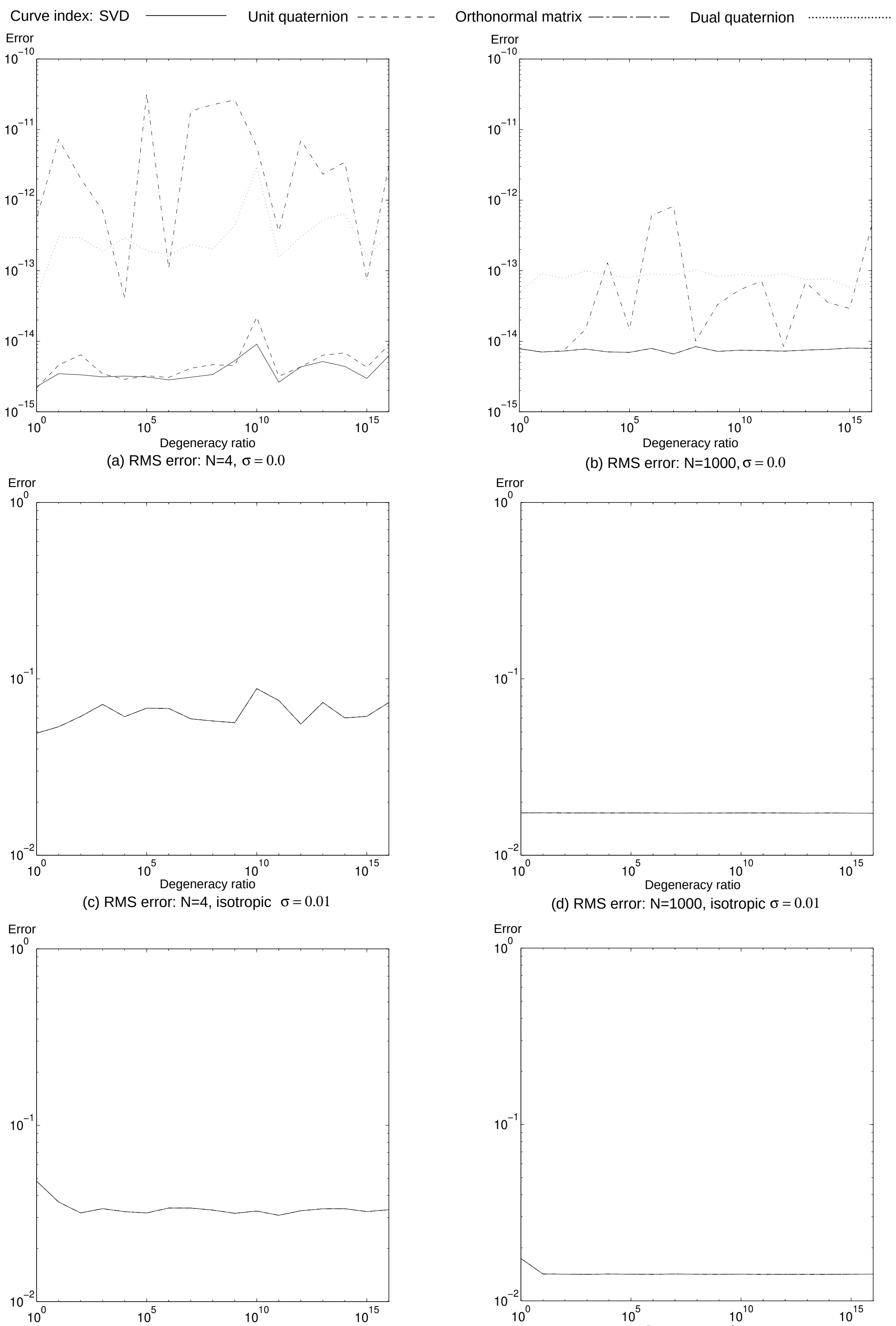


Figure 5: Graphs (log-log scale) of RMS errors of algorithms on 2-D degenerate data sets for zero noise and (a) N = 4, (b) N = 1000, isotropic noise with $\sigma = 0.01$ and (c) N = 4, (d) N = 1000, and anisotropic noise with $\sigma = 0.01$ and (e) N = 4, (f) N = 1000.

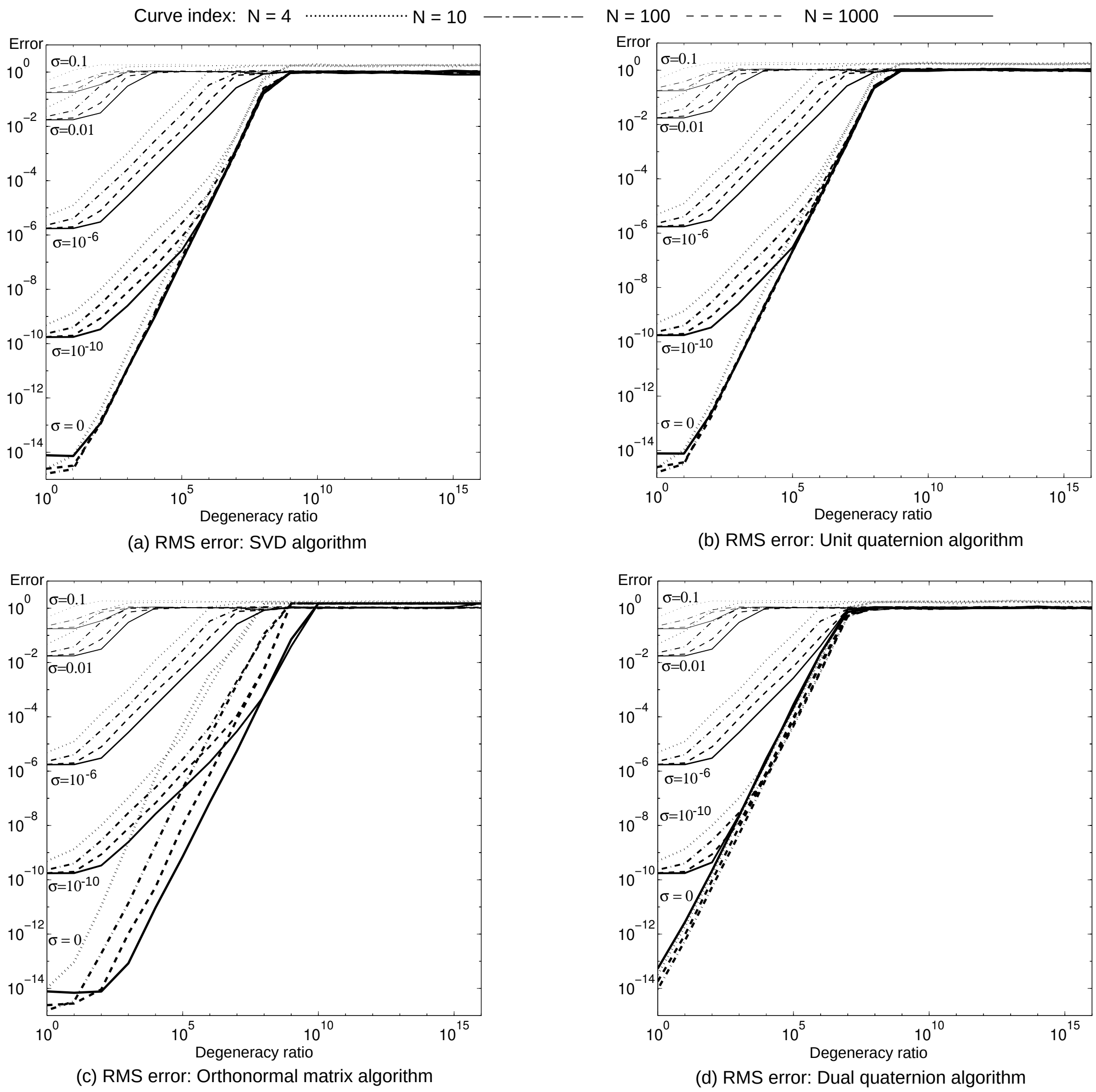
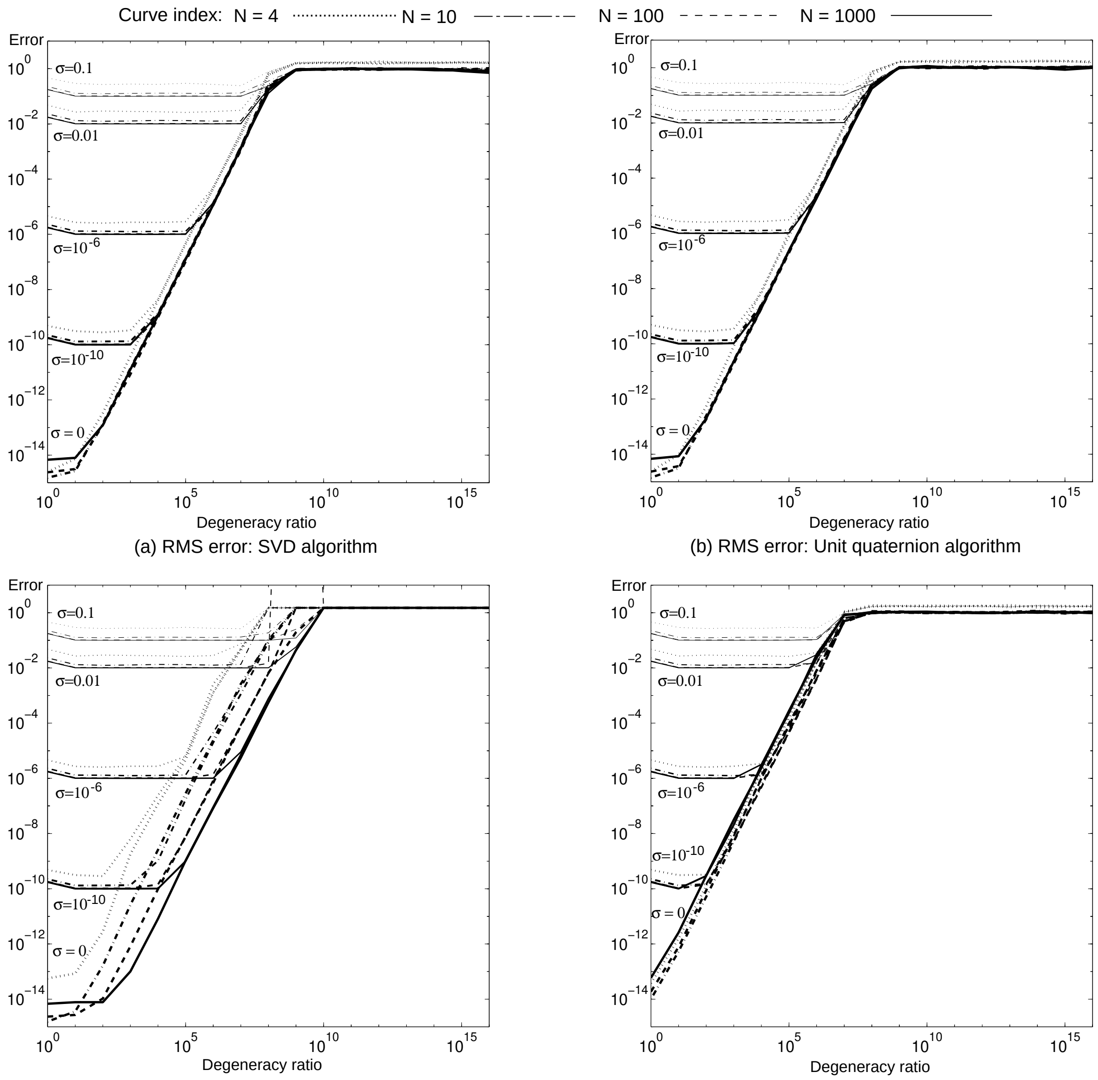


Figure 6: RMS errors of (a) SVD, (b) Unit quaternion, (c) Orthonormal matrix and (d) Dual quaternion algorithms for 1-D degenerate data sets under isotropic noise. Curves on graphs (log-log scale) present errors for four data set sizes, N = 4, 10, 100, 1000, and for five noise levels $\sigma = 0.0, 10^{-10}, 10^{-6}, 0.01, 0.1$. The thickest curves are $\sigma = 0$, while the thinnest are $\sigma = 0.1$.

than that. For anisotropic noise the error remains at the original level of noise until the numeric instability occurs, and then breakdown occurs steadily as the line diameter decreases.

Figure 8 shows the direct comparison of the methods under the extreme conditions. Under isotropic noise the algorithms behave similarly. Under anisotropic noise or no noise at all the differences in numeric stability become apparent, with better results achieved for larger data sets. The dual quaternion algorithm breaks down first in all cases. The SVD and unit quaternion methods are quite similar, with SVD performing marginally better. The orthonormal matrix algorithm's



(a) RMS error: SVD algorithm

(b) RMS error: Unit quaternion algorithm

(c) RMS error: Orthonormal matrix algorithm

(d) RMS error: Dual quaternion algorithm

Figure 7: RMS errors of (a) SVD, (b) Unit quaternion, (c) Orthonormal matrix and (d) Dual quaternion algorithms for 1-D degenerate data sets under anisotropic noise. Curves on graphs (log-log scale) present errors for four data set sizes, N = 4, 10, 100, 1000, and for five noise levels $\sigma = 0.0, 10^{-10}, 10^{-6}, 0.01, 0.1$. The thickest curves are $\sigma = 0$, while the thinnest are $\sigma = 0.1$.

performance is highly dependent on the data set size, actually performing the best for large data sets under anisotropic noise.

3.3.3 3-D to 0-D degeneracy

In this experiment the data set degenerates into a point. All three dimensions of the 3-D cube are steadily shrunk from their 2 unit size down to the level of machine precision at $2 * 10^{-16}$. Figures 9 and 10 give the response curves of the algorithms under isotropic and anisotropic noise,

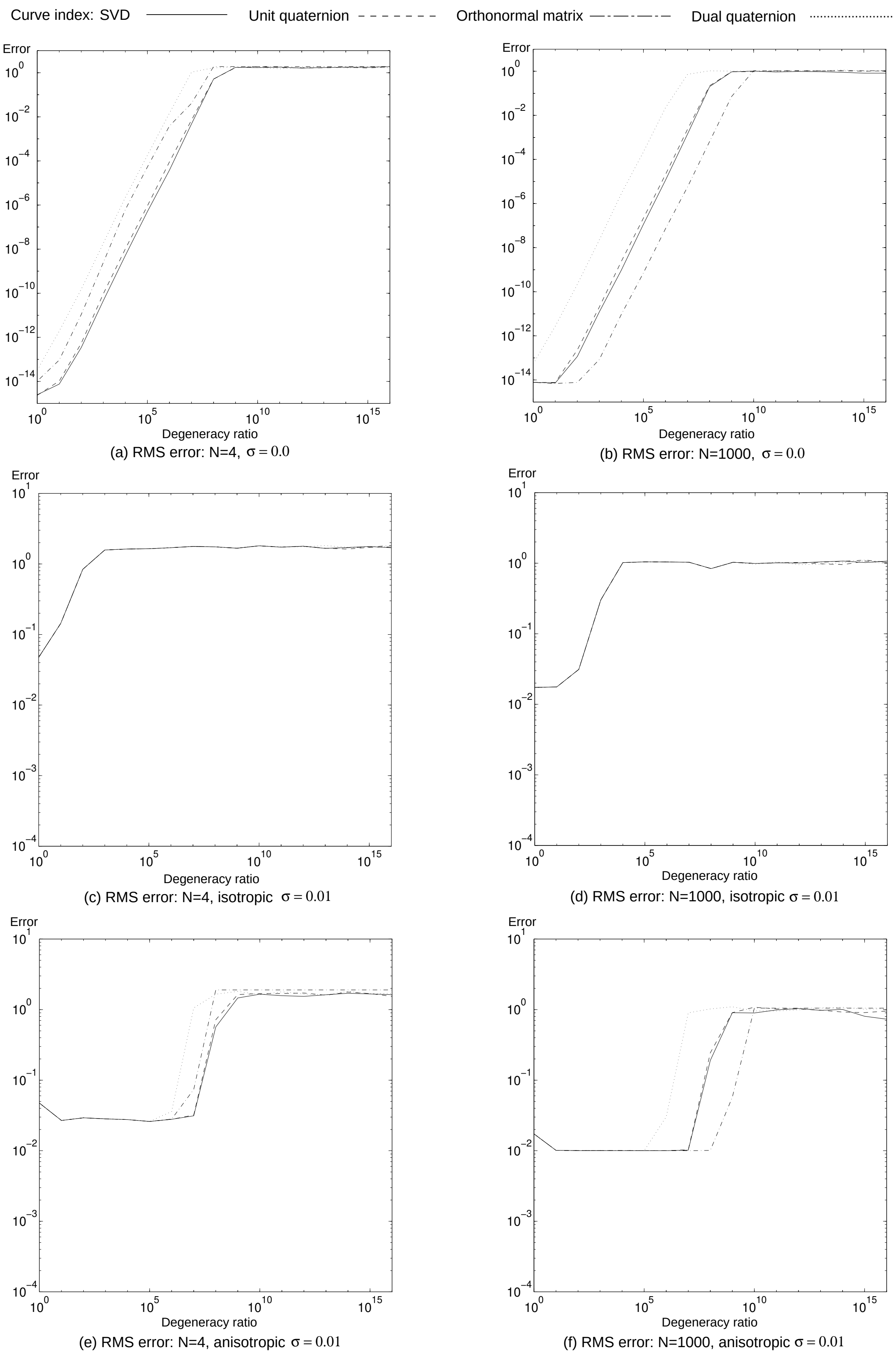


Figure 8: Graphs (log-log scale) of RMS errors of algorithms on 1-D degenerate data sets for zero noise and (a) N = 4, (b) N = 1000, isotropic noise with $\sigma = 0.01$ and (c) N = 4, (d) N = 1000, and anisotropic noise with $\sigma = 0.01$ and (e) N = 4, (f) N = 1000.

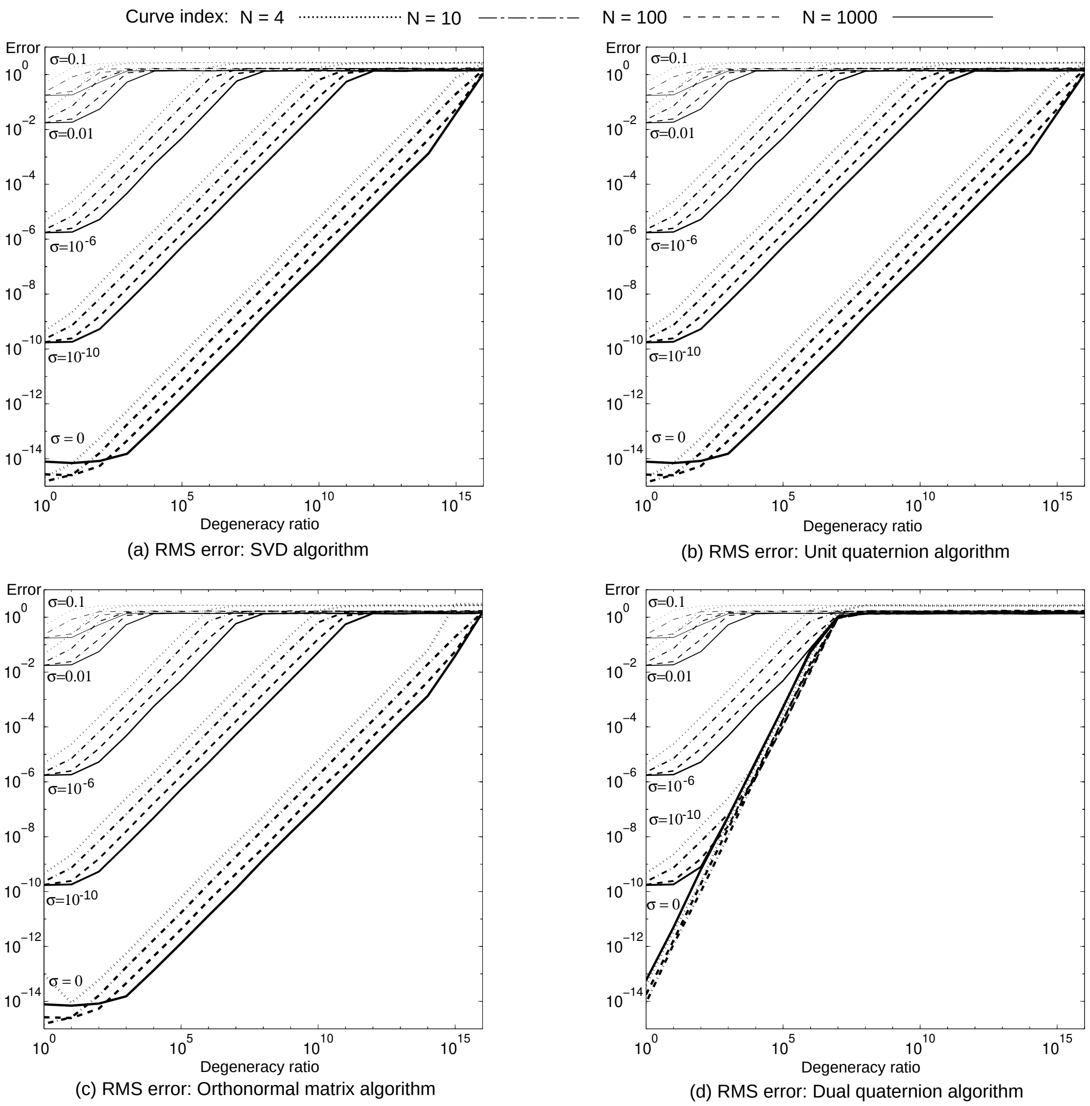


Figure 9: RMS errors of (a) SVD, (b) Unit quaternion, (c) Orthonormal matrix and (d) Dual quaternion algorithms for 0-D degenerate data sets under isotropic noise. Curves on graphs (log-log scale) present errors for four data set sizes, N = 4, 10, 100, 1000, and for five noise levels $\sigma = 0.0, 10^{-10}, 10^{-6}, 0.01, 0.1$. The thickest curves are $\sigma = 0$, while the thinnest are $\sigma = 0.1$.

respectively, for five levels of noise and four data set sizes. In the anisotropic case, the noise level is actually uniform, but reduced by the degeneracy ratio in all three dimensions.

As with the linear data set, there are too many unconstrained degrees of freedom in essentially matching a point to a point, meaning the algorithms will break down. The graphs in Figures 9 and 10 show the breakdown points of the algorithms, which are slightly different from the previous experiment. Under isotropic noise the same steady rise in error as the cube size decreases is seen. In this case total breakdown is achieved as the size of the cube reaches the noise level for all algorithms

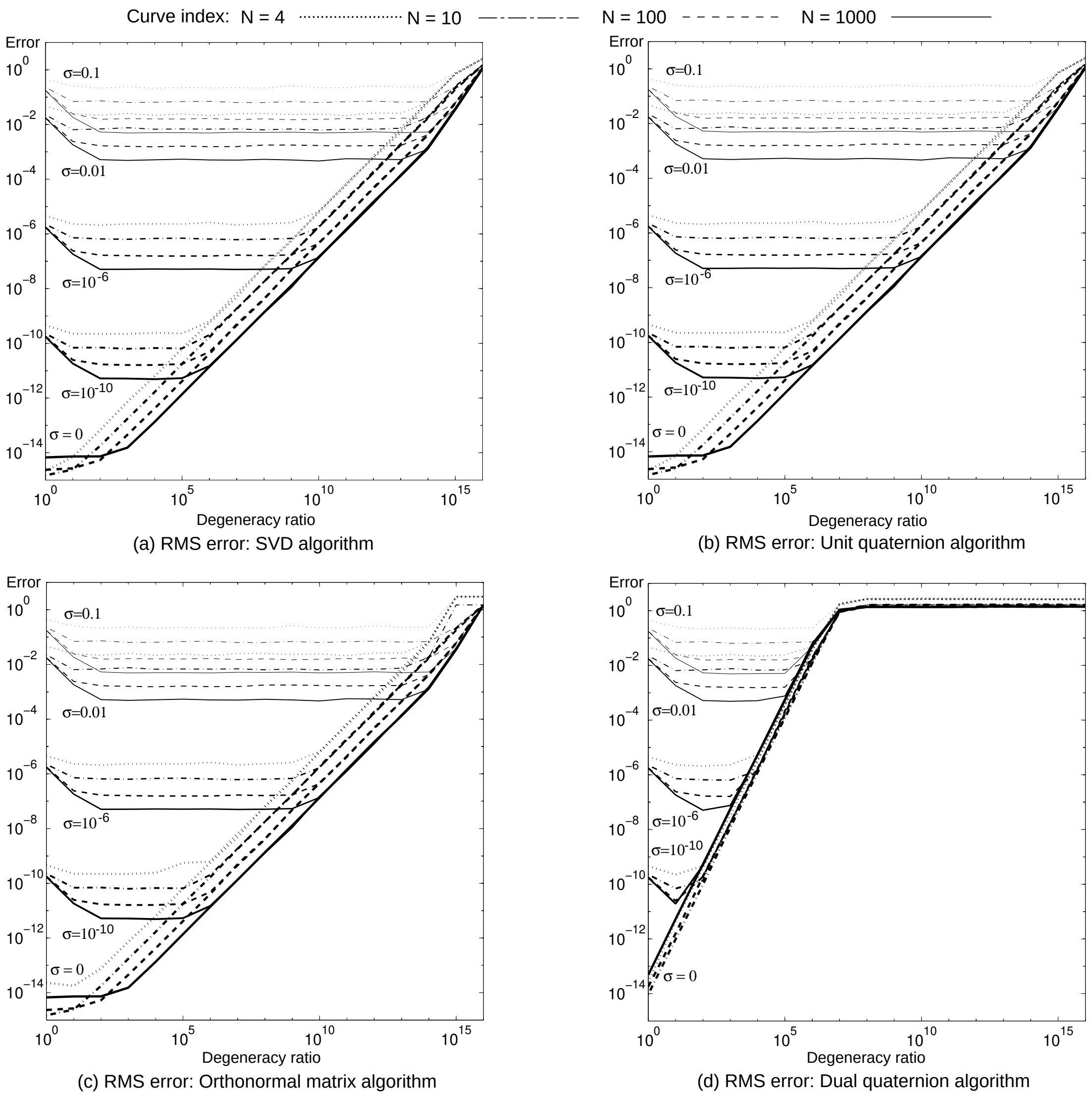


Figure 10: RMS errors of (a) SVD, (b) Unit quaternion, (c) Orthonormal matrix and (d) Dual quaternion algorithms for 0-D degenerate data sets under anisotropic noise. Curves on graphs (log-log scale) present errors for four data set sizes, N = 4, 10, 100, 1000, and for five noise levels $\sigma = 0.0, 10^{-10}, 10^{-6}, 0.01, 0.1$. The thickest curves are $\sigma = 0$, while the thinnest are $\sigma = 0.1$.

but the dual quaternion method. Its stability is apparently less than the others in this case, and breakdown occurs earlier under small amounts of noise. When the amount of noise is reduced as in the anisotropic model, the error response of the algorithms is consistent with the nominal noise level until there is effectively no noise and a common breakdown curve is joined. Again, the response of the dual quaternion algorithm is not as robust as the others.

The magnitude of the difference in stability between the dual quaternion algorithm and the others is apparent in the comparison graphs of Figure 11. Here it can be seen that breakdown

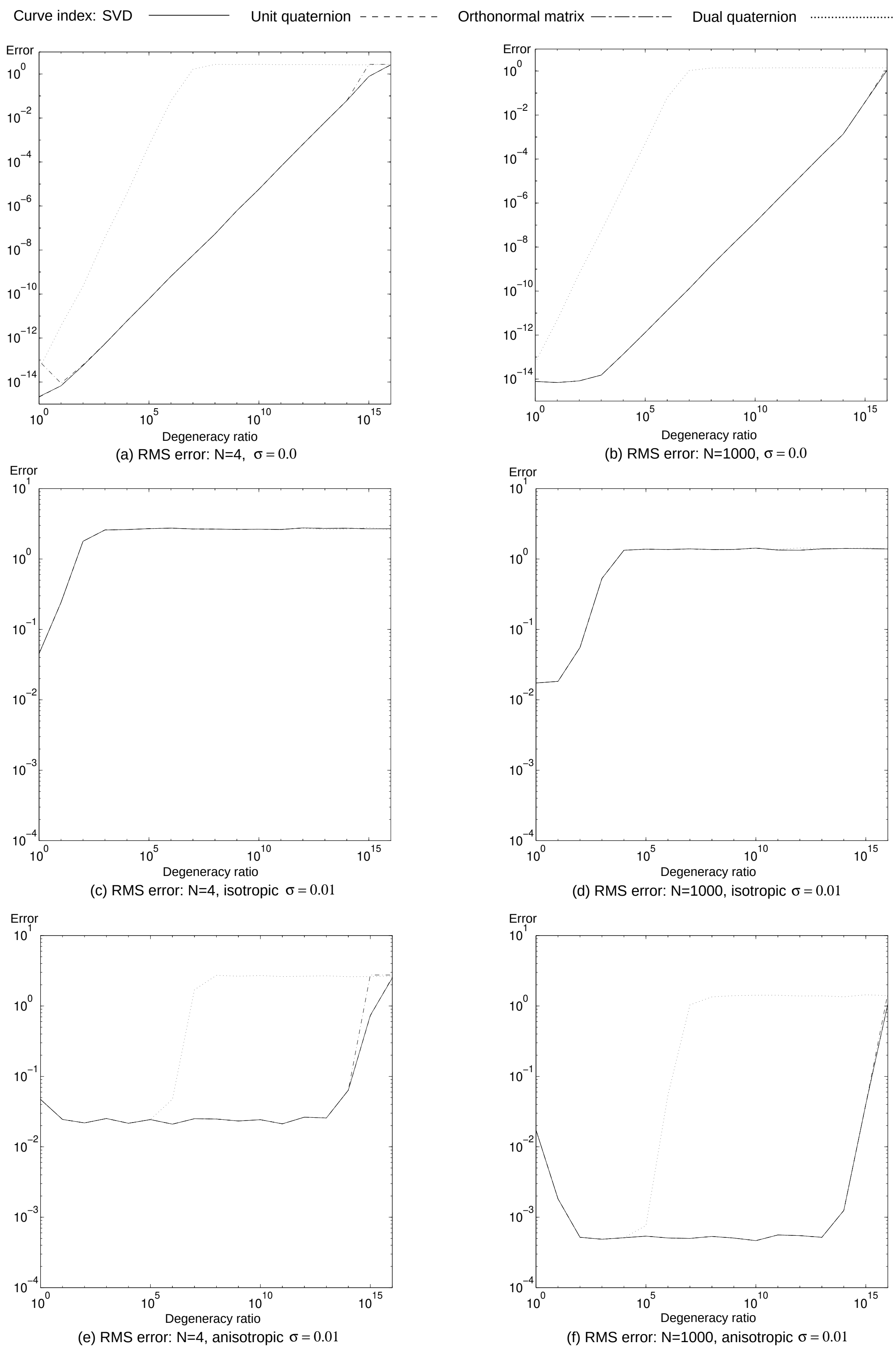


Figure 11: Graphs (log-log scale) of RMS errors of algorithms on 0-D degenerate data sets for zero noise and (a) N = 4, (b) N = 1000, isotropic noise with $\sigma = 0.01$ and (c) N = 4, (d) N = 1000, and anisotropic noise with $\sigma = 0.01$ and (e) N = 4, (f) N = 1000.

occurs much earlier, excepting in the cases of large levels of isotropic noise. The other responses are virtually identical, with the orthonormal matrix curves breaking down slightly sooner when a small data set size is used.

3.4 Efficiency experiments

From a theoretical standpoint, each of the algorithms examined has a time complexity of $O(N)$. In this section the actual coefficients of the linear time functions are compared. The execution times of the algorithms were recorded during all of the previous four experiments on accuracy and stability. All timings were performed a 50 MHz Sun Sparc Station 10. The routines were coded with efficiency in mind, and an optimizing compiler was used (one could probably slightly improve the following results with some substantial effort). The recorded times are listed in Table 1. These represent the zero noise cases for both nondegenerate 3-D data sets, as well as the degenerate 2-D, 1-D and 0-D sets. The timings observed under different levels of noise were virtually identical to those in the table and are not included here. This data is also graphed in Figure 12.

First consider the coefficient of the linear term of the time equation. All of the algorithms except the dual quaternion algorithm initially spend time computing the same set of sums that is the outer products of the corresponded point coordinates. Therefore the linear terms, which are dependent on this computation, should be identical for these three. The dual quaternion algorithm computes a different set of sums. The relative values of these two terms are shown in the slopes of the graphs in the rightmost column of Figure 12. It is apparent that the number of computations involved in the dual quaternion's summations is greater than that of the other three.

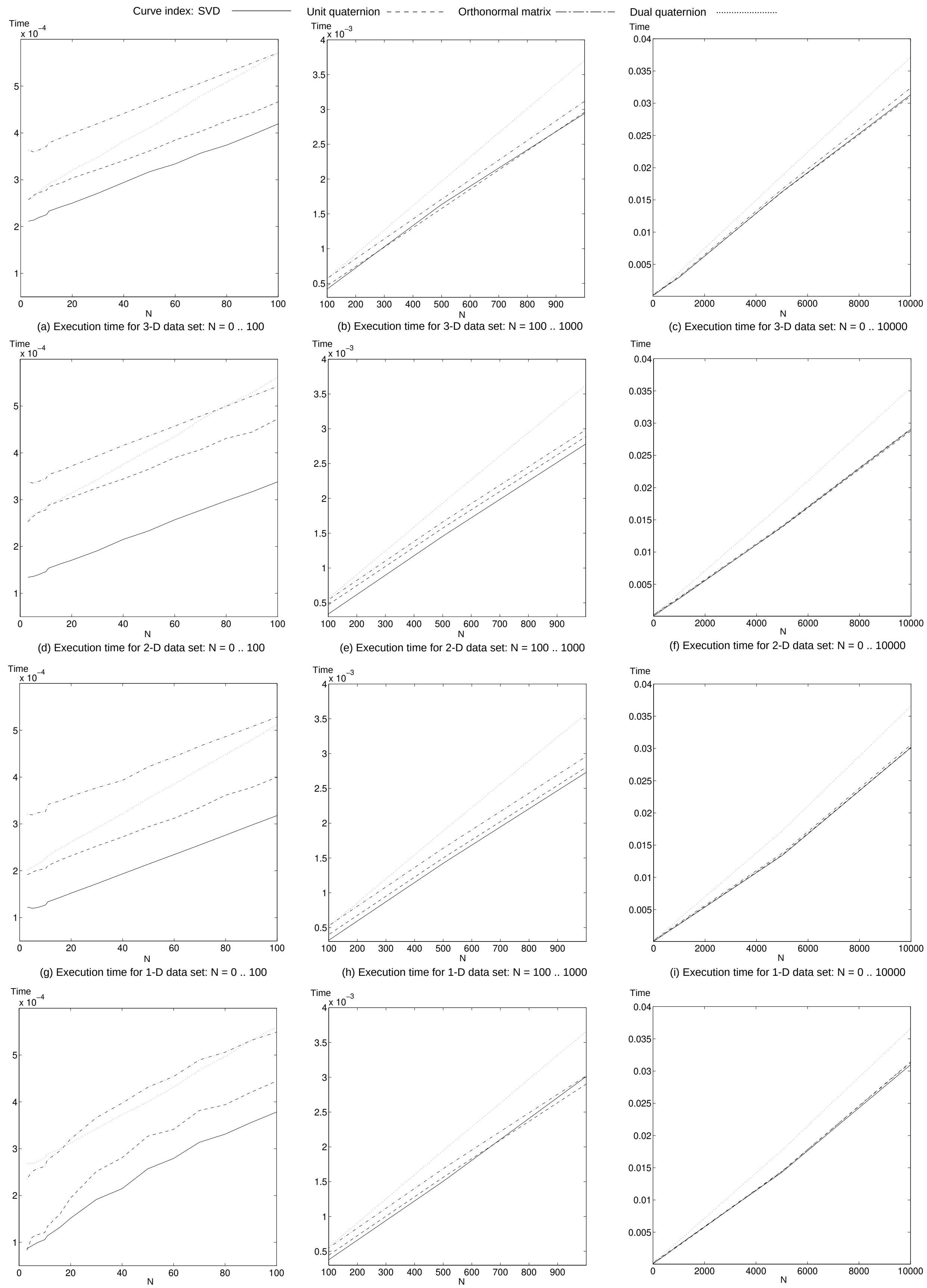
Next consider the constant terms of the time equations. Here each algorithm has a varying amount of time. The SVD algorithm computes the singular value decomposition of a 3×3 matrix, then performs matrix multiplication and determinant operations. The orthonormal matrix algorithm determines the eigensystem of a 3×3 matrix, in addition to a greater number of matrix operations. The unit quaternion algorithm computes the eigensystem of a 4×4 matrix, and a minimal amount of computation for representation conversion. Lastly, the dual quaternion algorithm also computes the eigensystem of a 4×4, as well as some matrix calculations. Additionally, each

Table 1: Execution times of SVD, Unit quaternion (UQ), Orthonormal matrix (OM) and Dual quaternion (DQ) algorithms on both nondegenerate and degenerate data sets. Times are listed in milliseconds and are the average response of 100 trials each.

	Nondegenerate 3-D point set				Degenerate 2-D point set			
N	SVD	UQ	OM	DQ	SVD	UQ	OM	DQ
3	0.212	0.257	0.364	0.258	0.134	0.253	0.338	0.257
5	0.214	0.267	0.359	0.264	0.136	0.264	0.335	0.268
10	0.225	0.278	0.371	0.286	0.146	0.278	0.345	0.283
20	0.250	0.304	0.399	0.321	0.171	0.305	0.372	0.314
50	0.317	0.362	0.464	0.411	0.234	0.365	0.436	0.406
100	0.419	0.466	**0.571**	**0.571**	0.338	0.472	**0.542**	**0.562**
500	**1.635**	**1.576**	1.709	1.966	1.455	1.573	1.661	1.931
1000	**2.942**	**2.961**	3.117	3.708	2.781	2.888	2.985	3.624
5000	**16.231**	**16.187**	16.626	18.716	**13.951**	**13.884**	14.090	17.486
10000	31.297	31.076	32.352	37.112	29.049	28.814	29.141	35.497
	Degenerate 1-D point set				Degenerate 0-D point set			
N	SVD	UQ	OM	DQ	SVD	UQ	OM	DQ
3	0.122	0.191	0.321	0.201	0.086	0.083	0.235	0.269
5	0.120	0.197	0.319	0.208	0.093	0.111	**0.251**	**0.269**
10	0.127	0.204	0.326	0.224	0.106	0.121	0.262	0.280
20	0.152	0.232	0.359	0.261	0.152	0.195	0.321	0.312
50	0.214	0.294	0.422	0.355	0.257	0.327	0.431	0.400
100	0.318	0.400	0.528	0.512	0.378	0.444	**0.549**	**0.560**
500	1.424	1.500	**1.641**	**1.889**	1.505	1.562	1.687	1.946
1000	2.731	2.804	2.960	3.575	**3.011**	**2.903**	3.022	3.660
5000	13.408	13.525	13.759	17.249	**14.285**	**14.395**	14.452	17.700
10000	30.143	30.181	30.591	36.559	30.963	**31.405**	**31.287**	36.580

of the algorithms performs a similar amount of work to compute the translation vector once the rotation matrix is known.

The relative magnitudes of the constant terms corresponding to these calculations are best seen at the beginning of the graphs in the leftmost column of Figure 12. Here, the SVD computation seems the most efficient, while determining the eigensystem of a 4×4 matrix and a minimal amount of matrix computation by the quaternion algorithms is faster than the additional matrix calculations of the orthonormal matrix method. Notice that this time is not constant across all of the data configurations. This is because the SVD and eigensystem routines used are iterative, and appear to be able to perform the calculations easier in the degenerate cases. Note that if the non-iterative algorithms had been used this would not be the case.



(a) Execution time for 3-D data set: N = 0 .. 100
(b) Execution time for 3-D data set: N = 100 .. 1000
(c) Execution time for 3-D data set: N = 0 .. 10000
(d) Execution time for 2-D data set: N = 0 .. 100
(e) Execution time for 2-D data set: N = 100 .. 1000
(f) Execution time for 2-D data set: N = 0 .. 10000
(g) Execution time for 1-D data set: N = 0 .. 100
(h) Execution time for 1-D data set: N = 100 .. 1000
(i) Execution time for 1-D data set: N = 0 .. 10000
(j) Execution time for 0-D data set: N = 0 .. 100
(k) Execution time for 0-D data set: N = 100 .. 1000
(l) Execution time for 0-D data set: N = 0 .. 10000

Figure 12: Execution times (in seconds) of algorithms on nondegenerate and degenerate data sets. Graphs are shown for data set sizes N = 4 .. 100, N = 100 .. 1000, and N = 4 .. 10000.

The SVD and eigensystem algorithms also show a dependence upon the data set size, although it is not as consistent as the dependence upon the data dimension. In several of the cases as the data set size grows, the SVD computation time increases causing the overall execution of this routine to be slightly slower than that of the unit quaternion. The crossover of these curves can be seen in the center column of Figure 12, and the corresponding entries are highlighted in Table 1. The variation of the eigensystem calculation does not seem as susceptible to data set size as the SVD.

So, overall, if the data set size is less than 100, the SVD algorithm is the quickest, by as much as 50%. For large data set sizes the relative difference of the SVD, unit quaternion and orthonormal matrix methods becomes negligible, less than 1%. But all three are superior to the dual quaternion technique, which is approximately 20% slower.

4 Conclusions

A comparison has been made between four algorithms which compute a rigid 3-D transformation between two sets of corresponding points in closed form: SVD, unit quaternion, orthonormal matrix and dual quaternion. As with most comparisons, no one algorithm was found to be superior in all cases. However, some final conclusions can be made.

The difference in accuracy of the algorithms on a nondegenerate 3-D point set, even for various noise levels, is almost insignificant. This is perhaps not unexpected since they were designed to solve the same problem. This conclusion is not in agreement with earlier findings of Walker, *et al.* in which it was stated "the two algorithms produce the same rotation errors ... for the translation errors, the DQ algorithm exhibits better performance than the SVD algorithm ... " [8, pg. 364]. It is not known how they derived their conclusion, but it is certainly not supported by the data here.

A greater separation of algorithm performance was noticed in the stability testing. Here, in most cases, the SVD and unit quaternion methods were very similar and usually the most stable. The orthonormal matrix method was not as stable for planar data sets, but was superior for certain degenerate data sets of large size. In none of the testing was the dual quaternion algorithm the most stable, and usually it was seen to break down before the others.

In terms of efficiency, for small data set sizes the SVD algorithm is by far the quickest. For larger data sets the difference in speed between all but the dual quaternion algorithm is not overly significant. The dual quaternion is significantly slower in these cases. In its defence, this algorithm was designed to include normal direction information in the solution. It is quite possible that if the other algorithms were suitably modified to include these extra constraints, then the dual quaternion's relative performance would improve.

Lastly, it might be wondered how the accuracy and stability of the algorithms is affected by incorrect point correspondences, or *outlier* data. In this case, all of the algorithms fall in the least squares category of solution, which is known not to be the optimal approach in these instances [4].

So, in conclusion, it appears that the SVD algorithm provides the best performance and speed for smaller data sets. In the case of larger data sets, a preference for a rotation representation, 3×3 matrix or unit quaternion, may alter your decision.

References

[1] Arun, K.S., Huang, T.S. and Blostein, S.D. "Least-Squares Fitting of Two 3-D Point Sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 5, pp. 698-700, 1987.

[2] Horn, B.K.P. "Closed-form Solution of Absolute Orientation using Unit Quaternions," *Journal of the Optical Society of America, Series A*, vol. 4, no. 4, pp. 629-642, 1987.

[3] Horn, B.K.P., Hilden, H.M. and Negahdaripour, S. "Closed-form Solution of Absolute Orientation Using Orthonormal Matrices," *Journal of the Optical Society of America, Series A*, vol. 5, no. 7, pp. 1127-1135, 1988.

[4] Meer, P., Mintz, D., Rosenfeld, A. and Kim, D.Y. "Robust Regression Methods for Computer Vision : A Review," *International Journal of Computer Vision*, vol. 6, no. 1, pp. 59-70, 1991.

[5] Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. NUMERICAL RECIPES in C: The Art of Scientific Computing, Second Edition, (Cambridge University Press, 1992).

[6] Sabata, B. and Aggarwal, J.K. "Estimation of Motion From a Pair of Range Images: A Review," *CVGIP: Image Understanding*, vol. 54, no. 3, pp. 309-324, 1991.

[7] Schonemann, P. "A Generalized Solution of the Orthogonal Procrustes Problem," *Psychometrika*, vol. 31, pp. 1-10, 1966.

[8] Walker, M.W., Shao, L. and Volz, R.A. "Estimating 3-D Location Parameters Using Dual Number Quaternions," *CVGIP: Image Understanding*, vol. 54, no. 3, pp. 358-367, 1991.