NIST Technical Note 2164

On Computing Elastic Shape Distances between Curves in $d-$dimensional Space

Javier Bernal
Jim Lawrence
Günay Doğan
Charles Hagwood

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2164

National Institute of
Standards and Technology
U.S. Department of Commerce

NIST Technical Note 2164

On Computing Elastic Shape Distances between Curves in $d-$dimensional Space

Javier Bernal
Information Technology Laboratory
Applied and Computational Mathematics Division

Jim Lawrence
George Mason University
Information Technology Laboratory
Applied and Computational Mathematics Division

Günay Doğan
Information Technology Laboratory
Applied and Computational Mathematics Division

Charles Hagwood
Information Technology Laboratory
Statistical Engineering Division

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2164

June 2021

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
James K. Olthoff, Performing the Non-Exclusive Functions and Duties of the Under Secretary of Commerce for Standards and Technology & Director, National Institute of Standards and Technology

National Institute of Standards and Technology Technical Note 2164
Natl. Inst. Stand. Technol. Tech. Note 2164, 39 pages (June 2021)
CODEN: NTNOEF

This publication is available free of charge from:
https://doi.org/10.6028/NIST.TN.2164

Abstract

The computation of the elastic registration of two simple curves in higher dimensions and therefore of the elastic shape distance between them has been investigated by Srivastava et al. Assuming the first curve has one or more starting points, and the second curve has only one, they accomplish the computation, one starting point of the first curve at a time, by minimizing an L^2 type distance between them based on alternating computations of optimal diffeomorphisms of the unit interval and optimal rotation matrices that reparametrize and rotate, respectively, one of the curves. We recreate the work by Srivastava et al., but in contrast to it, again for curves in any dimension, we present a Dynamic Programming algorithm for computing optimal diffeomorphisms that is linear, and justify in a purely algebraic manner the usual algorithm for computing optimal rotation matrices, the Kabsch-Umeyama algorithm, which is based on the computation of the singular value decomposition of a matrix. In addition, we minimize the L^2 type distance with a procedure that alternates computations of optimal diffeomorphisms with successive computations of optimal rotation matrices for all starting points of the first curve. Carrying out computations this way is not only more efficient all by itself, but, if both curves are closed, allows applications of the Fast Fourier Transform for computing successively in an even more efficient manner, optimal rotation matrices for all starting points of the first curve.

Key words

dynamic programming; elastic shape distance; FFT; rotation matrix; shape analysis; singular value decomposition.

Table of Contents

List of Figures

1. Introduction

In this paper, following ideas in [13, 14], we address the problem of computing the elastic shape distance between two simple curves, not necessarily closed, in $d-$dimensional space, d a positive integer, or equivalently the problem of computing the elastic registration of two such curves. If neither curve is closed so that each curve has a fixed starting point, this is done by finding a diffeomorphism of the unit interval, and a $d \times d$ rotation matrix, that reparametrizes and rotates, respectively, one of the curves, not necessarily the same curve for both operations, so that an L^2 type distance between the curves is minimized. The resulting minimum distance is then the elastic shape distance between the curves and the result of the reparametrization and rotation of the curves is their elastic registration. On the other hand, if, say, the first curve is closed so that any point in it can be treated as a starting point of the curve, then a finite subset of consecutive points is selected in the curve in the direction in which the curve is defined, in such a way that the subset is reasonably dense in the curve (by joining consecutive points in the subset with line segments, the resulting piecewise linear curve should be a reasonable approximation of the curve). This finite subset of the first curve is interpreted to be the set of starting points of the curve. A fixed starting point is then identified on the other curve, the second curve, perhaps arbitrarily if the curve is closed. In [13, 14], given a point in the set above interpreted to be the set of starting points of the first curve, using the point as the starting point of the (first) curve, an optimal diffeomorphism and an optimal rotation matrix are found in the same manner as described above for the case in which neither curve is closed. Again in [13, 14], this is done for each point in the set, and the point for which the L^2 type distance is the smallest is then considered to be the optimal starting point in the first curve, and the optimal diffeomorphism and optimal rotation matrix associated with it are then treated as the optimal diffeomorphism and optimal rotation matrix that produce the elastic registration of the two curves and the elastic shape distance between them.

We note that above we have tacitly assumed that the curves are defined in the proper directions for the purpose of unambiguously comparing their shapes. We have done this for simplicity as in reality it may not be the case. Given a simple curve in $d-$dimensional space, it has two possible directions in which it can be defined. In particular, in $2-$dimensional space (the plane), a closed simple curve is defined in either the clockwise direction or the counterclockwise direction, and if the shapes of two closed simple curves in the plane are to be compared, it only makes sense that both be defined in the same direction (clockwise or counterclockwise). Unfortunately, in general, defining two curves in $d-$dimensional space in the proper directions cannot be done this way, and the only alternative is first to compute the elastic shape distance and registration with the curves as given and then reverse the direction of one of the curves and do the computations again. The smaller of the two computed elastic shape distances then determines the proper directions of the curves, and therefore their correct elastic registration. Again for simplicity, in the rest of this paper, given two simple curves in $d-$dimensional space, we assume they are defined in the proper directions for all purposes, keeping in mind that if this is not the case,

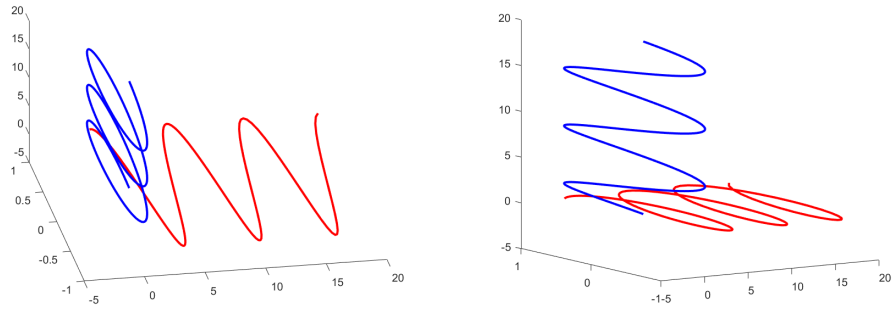

Fig. 1. Two views of the same two helices, curves in 3−d space. The positive $z-$axis is the axis of rotation of one helix, while the positive $x-$axis is the axis of rotation of the other one. Their shapes are essentially identical thus the elastic shape distance between them should be essentially zero.

all that is required to fix them, is that the direction of one of the curves be reversed.

Being able to compute the elastic registration of two curves and the elastic shape distance between them in higher dimensions, in particular in three dimensions, is useful for studying fiber tracts, geological terrains, protein structures, facial surfaces, color images, cylindrical helices, etc. See Figure 1 that depicts two such curves in 3−dimensional space. (Note that in the plots there, the $y-$axis is not to scale relative to the $x-$axis and the $z-$axis).

As mentioned above, in this paper, we address the problem of computing the elastic registration of two simple curves in $d-$dimensional space, and the elastic shape distance between them following ideas in [13, 14]. However, in contrast to [13, 14], we present a method for computing optimal diffeomorphisms that is linear, and justify in a purely algebraic manner the usual algorithm for computing optimal rotation matrices. With the convention that if at least one of the curves is closed, the first curve is closed, so that any point in it can then be treated as a starting point of the curve, we redefine the L^2 type distance to allow for the second curve to be reparametrized while the first one is rotated, and select, in the appropriate manner, just as it is done in [13, 14], a finite subset of points in the (first) curve (one point if neither curve is closed) which we interpret to be the set of starting points of the curve. We then minimize the redefined L^2 type distance with an iterative procedure that alternates computations of optimal diffeomorphisms (a constant number of them per iteration for reparametrizing the second curve) with successive computations of optimal rotation matrices (for rotating the first curve) for all starting points of the first curve. (Note that in [13, 14] the alternating computations occur one starting point of the first curve at a time). As noted in [4], carrying out computations this way is not only more efficient all by itself, but, if both curves are closed, allows applications of the Fast Fourier Transform (FFT) as demonstrated in [5] for $d = 2$, for computing successively in an even more efficient manner, optimal rotation matrices for all starting points of the first curve.

In Section 2 of this paper, we describe a fast linear Dynamic Programming (DP) algorithm for computing an approximately optimal diffeomorphism for the elastic registration

of two simple curves in $d-$dimensional space, the curves not necessarily closed, each curve with a fixed starting point, the computation of the registration based only on reparametrizations (with diffeomorphisms of the unit interval) of one of the curves. In Section 3, we describe and justify the usual algorithm, the Kabsch-Umeyama algorithm, for computing an approximately optimal $d \times d$ rotation matrix for the rigid alignment of two simple curves in $d-$dimensional space, the curves not necessarily closed, each curve with a fixed starting point, the computation of the alignment based only on rotations (with rotation matrices) of one of the curves. The algorithm, which is based on the computation of the singular value decomposition of a matrix, is justified in a purely algebraic manner. In Section 4, given two simple curves in $d-$dimensional space, one considered to be the first curve, the other the second curve, and then keeping in mind that the first curve can have one or more starting points while the second curve has only one, we redefine the L^2 type distance between them as hinted above, and present the iterative procedure mentioned above that minimizes this distance by alternating computations of approximately optimal diffeomorphisms (a constant number of them per iteration) with successive computations of approximately optimal rotation matrices for all starting points of the first curve. In Section 5, assuming both curves are closed, we show how the FFT can be used to speed up the successive computations of approximately optimal rotation matrices for all starting points of the first curve. Finally, in Section 6, we present results computed with implementations of our methods applied on curves in $3-$d space of the helix and spherical ellipsoid kind.

2. Computation of Diffeomorphism for Registration of Curves

In this section, we describe a fast Dynamic Programming (DP) algorithm that runs in linear time to compute an approximately optimal diffeomorphism for the elastic registration of two simple curves in $d-$dimensional space, d a positive integer, the curves not necessarily closed, each curve with a fixed starting point. Here the computation of the registration is based only on reparametrizations (with diffeomorphisms) of one of the curves. Because the algorithm depends on a couple of input parameters that should be small but sometimes are too small, it is not guaranteed to succeed every time, but in our experiments we have observed very convincing results for every problem on which the algorithm was applied, a few times after adjustments to the parameters. Here and in what follows, given $T > 0$ and an integer $N > 0$, we say a curve is discretized by a partition $\{t_i\}_{i=1}^N$, $t_1 = 0 < t_2 < \ldots < t_N = T$, of $[0,T]$, if a function $\beta : [0,T] \to \mathbb{R}^d$ has been identified to represent the curve (β is continuous and satisfies that its range is exactly the curve), and the curve is given as the set of N nodes equal to $\{\beta(t_i), i = 1,\ldots,N\}$. On input, the two curves under consideration are given as discrete sets of nodes in the curves, the result of discretizing the curves by partitions not necessarily uniform of $[0,1]$, the numbers of nodes in the curves not necessarily equal. Given that the numbers of nodes in the curves are N and M, respectively, then the algorithm is indeed linear as it runs in $O(N+M)$ time (see below). We note that what follows in this section about the algorithm, already appears in [1] for $d = 2$. We repeat it not only for self-containment but for clarity as it is what must be said about the algorithm

for any d besides $d = 2$.

Assume the curves can be represented by functions $\beta_n : [0,1] \to \mathbb{R}^d,\ n = 1,2$, that are absolutely continuous (see [2, 13]) and of unit length, and given n, $n = 1,2$, assume $\beta_n(0)$ is the fixed starting point of β_n, and if β_n is closed, then $\beta_n(0) = \beta_n(1)$, $\dot{\beta}_n(0) = \dot{\beta}_n(1)$ (here and in what follows, we may refer to the curve that β_n represents simply by β_n). With $\|\cdot\|$ as the $d-$dimensional Euclidean norm, we define $q_n : [0,1] \to \mathbb{R}^d,\ n = 1,2$, by $q_n(t) = \dot{\beta}_n(t)/\|\dot{\beta}_n(t)\|^{1/2}$ ($d-$dimensional 0 if $\dot{\beta}_n(t)$ equals $d-$dimensional 0), q_n the shape function or square-root velocity function (SRVF) of β_n, and note that because β_n is of unit length, then $\int_0^1 \|q_n(t)\|^2 dt = 1 < \infty$, the integral here and in what follows computed as a Lebesgue integral, so that q_n is square integrable for $n = 1,2$ (see [2, 13]). Note as well that for $d = 1$, $q_n(t)$ equals $\text{sign}(\dot{\beta}_n(t))\sqrt{|\dot{\beta}_n(t)|}$, $n = 1,2$, the square-root slope function (SRSF) of β_n, and for any d, any real number C and any square-integrable $q : [0,1] \to \mathbb{R}^d$ with $\int_0^1 \|q(t)\|^2 dt = 1$, the function $\beta : [0,1] \to \mathbb{R}^d$ defined by $\beta(t) = C + \int_0^t q(s)|q(s)|ds$ is absolutely continuous and of unit length with SRVF equal to q almost everywhere on $[0,1]$ (see [2, 13]). Finally, we note, given an absolutely continuous function $\beta : [0,1] \to \mathbb{R}^d$ and γ a diffeomorphism of $[0,1]$ onto itself with $\gamma(0) = 0, \gamma(1) = 1, \dot{\gamma} > 0$, then $(\beta \circ \gamma)(0) = \beta(0)$, and if q is the SRVF of β, then the SRVF of $\beta \circ \gamma$ equals $(q \circ \gamma)\sqrt{\dot{\gamma}}$ almost everywhere on $[0,1]$ (see [2, 13]). Defining $F(t, \gamma(t), \dot{\gamma}(t)) = \|q_1(t) - \sqrt{\dot{\gamma}(t)}q_2(\gamma(t))\|^2$, γ as above, we minimize the following energy with respect to γ

$$E(\gamma) = \int_0^1 F(t, \gamma(t), \dot{\gamma}(t))dt. \tag{1}$$

In practice, we need to solve a discretized version of the problem. Thus, for positive integers N, M, not necessarily equal, and partitions of $[0,1]$, $\{t_i\}_{i=1}^N, t_1 = 0 < t_2 < \ldots < t_N = 1$, $\{z_j\}_{j=1}^M, z_1 = 0 < z_2 < \ldots < z_M = 1$, not necessarily uniform, we assume β_1 and β_2 are given as lists of N and M points or nodes in the curves, respectively, where for $i = 1,\ldots,N$, $\beta_1(t_i)$ is the i^{th} point in the list for β_1, and for $j = 1,\ldots,M$, $\beta_2(z_j)$ is the j^{th} point in the list for β_2. We note that $\{\beta_1(t_i), i = 1,\ldots,N\}$ is then a set of consecutive points in β_1 in the direction in which β_1 is defined, and it should be defined in such a way that it is reasonably dense in β_1 (by joining consecutive points in it with line segments, the resulting piecewise linear curve should be a reasonable approximation of β_1). Similarly for $\{\beta_2(z_j), j = 1,\ldots,M\}$ with respect to β_2. We also assume $\dot{\beta}_1(t_i),\ i = 1,\ldots,N$, and $\dot{\beta}_2(z_j),\ j = 1,\ldots,M$, are approximately computed with centered finite differences from the lists of points for β_1 and β_2, respectively, so that $q_1(t_i),\ i = 1,\ldots,N$, and $q_2(z_j),\ j = 1,\ldots,M$, are then approximately computed by setting $q_1(t_i) = \dot{\beta}_1(t_i)/\|\dot{\beta}_1(t_i)\|^{1/2}$ ($d-$dimensional 0 if $\dot{\beta}_1(t_i)$ equals $d-$dimensional 0), $i = 1,\ldots,N$, and $q_2(z_j) = \dot{\beta}_2(z_j)/\|\dot{\beta}_2(z_j)\|^{1/2}$ ($d-$dimensional 0 if $\dot{\beta}_2(z_j)$ equals $d-$dimensional 0), $j = 1,\ldots,M$. Finally, given γ, treating (1) as a Riemann integral, we discretize (1) with the trapezoidal rule:

$$E(\vec{\gamma}) = \frac{1}{2}\sum_{i=1}^{N-1} h_i(F(t_{i+1}, \gamma_{i+1}, \dot{\gamma}_{i+1}) + F(t_i, \gamma_i, \dot{\gamma}_i)), \tag{2}$$

where $h_i = t_{i+1} - t_i$ for $i = 1, \ldots, N-1$, $F(t_i, \gamma_i, \dot{\gamma}_i)$ is understood to be $\|q_1(t_i) - \sqrt{\dot{\gamma}_i} q_2(\gamma_i)\|^2$ for $i = 1, \ldots, N$, $\vec{\gamma} = (\gamma_i)_{i=1}^N$, $\gamma_1 = 0, \gamma_N = 1$, $\gamma_i = \gamma(t_i)$, $\dot{\gamma}_i = (\gamma_{i+1} - \gamma_i)/h_i$ for $i = 1, \ldots, N-1$, $\dot{\gamma}_N = \dot{\gamma}_1$, and $q_2(\gamma_i)$, $i = 1, \ldots, N$, are approximations of q_2 at each γ_i obtained from the interpolation of $q_2(z_j)$, $j = 1, \ldots, M$, by a cubic spline. Thus, the problem of minimizing (1) with respect to γ then becomes, in practice, the problem of minimizing (2) with respect to a discretized γ.

For this purpose, we consider the $N \times M$ grid on the unit square with grid points labeled (i, j), i, j integers, $1 \leq i \leq N$, $1 \leq j \leq M$, each grid point (i, j) coinciding with the planar point (t_i, z_j).

If the mesh of each partition, i.e., $\max(t_{m+1} - t_m), 1 \leq m \leq N-1$, and $\max(z_{m+1} - z_m), 1 \leq m \leq M-1$, is sufficiently small, then the set of diffeomorphisms γ of $[0,1]$ onto itself with $\gamma(0) = 0, \gamma(1) = 1, \dot{\gamma} > 0$, can be approximated by the set of homeomorphisms of $[0,1]$ onto itself whose graphs are piecewise linear paths from grid point $(1,1)$ to grid point (N,M) with grid points as vertices, each linear component of a path having positive slope. We refer to the latter set as Γ. Then γ in Γ is an approximate diffeomorphism of $[0,1]$ onto itself and as such an energy conceptually faithful to (2) can be defined and computed for it. This is done one linear component of the graph of γ at a time.

Accordingly, given grid points (k,l), (i,j), $k < i$, $l < j$, that are endpoints of a linear component of the graph of γ, an energy of a trapezoidal nature over the line segment joining (k,l) and (i,j) is defined as follows:

$$E_{(k,l)}^{(i,j)} \equiv \frac{1}{2} \sum_{m=k}^{i-1} (t_{m+1} - t_m)(F_{m+1} + F_m), \tag{3}$$

$$F_m \equiv F(t_m, \alpha(t_m), L), \, m = k, \ldots, i,$$

where $F(t_m, \alpha(t_m), L)$ is understood to be $\|q_1(t_m) - \sqrt{L} q_2(\alpha(t_m))\|^2$ for $m = k, \ldots, i$, where α is the linear function from $[t_k, t_i]$ onto $[z_l, z_j]$ whose graph is the line segment, $\alpha(t_k) = z_l$, $\alpha(t_i) = z_j$, L is the slope of the line segment, and $q_2(\alpha(t_m)), m = k \ldots, i$, are approximations of q_2 at each $\alpha(t_m)$ obtained again from the interpolation of $q_2(z_r)$, $r = 1, \ldots, M$, by a cubic spline. Note, $L = \frac{z_j - z_l}{t_i - t_k} > 0$ as $z_j > z_l, t_i > t_k$. The energy for γ, which we denote by $E_0(\gamma)$, is then defined as the sum of the energies over the linear components of the graph of γ with α in (3) coinciding with γ on each component. Thus, the problem of minimizing (2) is then replaced by the approximately equivalent and easier to solve problem of minimizing $E_0(\gamma)$ for $\gamma \in \Gamma$.

For the purpose of efficiently computing $\gamma^* \in \Gamma$ of approximately minimum energy, i.e., $\gamma^* \in \Gamma$ such that $\gamma = \gamma^*$ approximately minimizes $E_0(\gamma)$ for $\gamma \in \Gamma$, we present an algorithm below that uses DP on sets of grid points around graphs of estimates of γ^*, one set at a time, each set containing the corner grid points $(1,1)$ and (N,M), each set defined each time a new estimate of γ^* is identified, each set coarser than the previous one, each set contained in and associated with an essentially thin region, i.e., a strip, in the unit square that contains the current estimate of γ^*. In this algorithm, a general DP procedure, Procedure *DP*, whose outline follows, is executed, for each strip as just described, on the set R of grid points

associated with the strip, for the purpose of minimizing $E_0(\gamma)$ (adjusted for R, see below) for $\gamma \in \Gamma$, γ satisfying that all of its vertices are in R. For such sets R the computational cost is low (the search space is relatively small), and their selection is such that it is highly likely the final DP solution is of approximately minimum energy and therefore can be assumed to be the desired γ^*. Since the collection of strips has the appearance of one single strip whose shape evolves as it mimics the shapes of graphs of estimates of γ^*, we think of the collection as indeed being one single strip, a dynamic strip that we call *adapting* strip accordingly. Thus, we propose using an adapting strip as just described with a width that is constant ($O(1)$) as it evolves around graphs of estimates of γ^*. Obviously we do not know γ^*, but can estimate it using DP solutions as the sets R associated with the strips become coarser. However, before going into the specifics of our proposed algorithm, we will describe Procedure *DP* operating on a generic R.

The set R of labeled grid points can be any subset of the interior grid points plus the corner grid points $(1,1)$, (N,M). Given any such R, we denote by $\Gamma(R)$ the set of elements of Γ with all vertices in R (note, R can have as few as three points in which case $\Gamma(R)$ has only one element). Accordingly, with the energy in (3) adjusted for R (see below) and $E_0(\gamma)$ for $\gamma \in \Gamma(R)$ adjusted accordingly, given a positive integer *layrs* (e.g., $layrs = 5$) which determines the size of certain neighborhoods to be searched (see below), then, based on DP, Procedure *DP* that follows, in $O(|R|)$ time, will often (depending on *layrs*) compute $\gamma^* \in \Gamma(R)$ such that $\gamma = \gamma^*$ approximately minimizes $E_0(\gamma)$ (adjusted for R, see below) for $\gamma \in \Gamma(R)$, $|R|$ the cardinality of R.

As the DP procedure progresses over the indices (i,j) in R, it examines function values on indices (k,l) in a trailing neighborhood $N(i,j)$ of (i,j). In the full DP, we would be examining all (k,l) in R, $1 \leqslant k < i, 1 \leqslant l < j$. This has high computational cost, and is not necessary for our applications. Using a much smaller square neighborhood $N(i,j)$ of ω points ($\omega = layrs$) per side gives satisfactory results. Thus, for each (i,j) in R, we examine at most ω^2 indices (k,l) in the trailing neighborhood $N(i,j)$ (defined below). Then the overall time complexity is $O(\omega^2|R|)$. We formally define $N(i,j)$ by

$$N(i,j) = \{(k,l) \in R : k \text{ is one of } \omega \text{ largest indices } < i \\ \text{and } l \text{ is one of } \omega \text{ largest indices } < j\}.$$

Note that in the case in which $N(i,j)$ happens to be empty then a grid point (k,l) in R, $k < i$, $l < j$, perhaps $(k,l) = (1,1)$, is identified and $N(i,j)$ is set to $\{(k,l)\}$

The DP procedure follows. First, however, we clarify some implicit conventions in the procedure logic. Grid points (i,j) in R are processed one at a time. However, the main loop in the DP procedure takes place over the single index i. We process index i in increasing order, and for each i, each grid point (i,j) in R is processed before moving to the next i. Also in the procedure, pairs of indices m_1, m_2 are retrieved from an index set $\mathscr{M}$, satisfying $m_1 < m_2$ with no other index in $\mathscr{M}$ greater than m_1 and less than m_2. This has the effect of adjusting for R energies computed according to (3).

procedure *DP*
  $E(1,1) = 0$
  for each $(i,j) \neq (1,1)$ in R in increasing order of i **do**
    for each $(k,l) \in N(i,j)$ **do**
      $\alpha =$ linear function, $\alpha(t_k) = z_l$, $\alpha(t_i) = z_j$
      $L =$ slope of line segment $\overline{(k,l)(i,j)}$
      $\mathscr{M} = \{m : k \leq m \leq i, \exists (m,n) \in R\}$
      $F_m = F(t_m, \alpha(t_m), L)$ for each $m \in \mathscr{M}$
      $E^{(i,j)}_{(k,l)} = \frac{1}{2}\sum_{m_1,m_2 \in \mathscr{M}} (t_{m_2} - t_{m_1})(F_{m_2} + F_{m_1})$
    end for
    $E(i,j) = \min_{(k,l)\in N(i,j)}(E(k,l) + E^{(i,j)}_{(k,l)})$
    $P(i,j) = \arg\min_{(k,l)\in N(i,j)}(E(k,l) + E^{(i,j)}_{(k,l)})$
  end for
end procedure

We note that $\gamma^* \in \Gamma(R)$ such that $\gamma = \gamma^*$ approximately minimizes $E_0(\gamma)$ (adjusted for R) for $\gamma \in \Gamma(R)$ can then be obtained by backtracking from (N,M) to $(1,1)$ with pointer P above. Accordingly, Procedure *opt-diffeom* that follows, will produce γ^* in the form $\vec{\gamma}^* = (\gamma^*_m)_{m=1}^N = (\gamma^*(t_m))_{m=1}^N$:

procedure *opt-diffeom*
  $\gamma^*_N = 1$
  $(i,j) = (N,M)$
  while $(i,j) \neq (1,1)$ **do**
    $(k,l) = P(i,j)$
    $\gamma^*_k = z_l$
    for each integer $m, k < m < i$ **do**
      $\gamma^*_m = \frac{(t_i - t_m)}{(t_i - t_k)} z_l + \frac{(t_m - t_k)}{(t_i - t_k)} z_j$
    end
    $(i,j) = (k,l)$
  end while
end procedure

In what follows, we present the linear DP algorithm which we call *adapt-DP*, based on DP restricted to an adapting strip, to compute an approximately optimal diffeomorphism for the elastic registration of two curves in $d-$dimensional space. It has parameters *layrs*, *lstrp*, set to small positive integers, say 5, 30, respectively. Parameter *layrs* is as previously described, while *lstrp* is an additional parameter that determines the width of the adapting strip (see below). Although *adapt-DP* is not guaranteed to be always successful (one or both of *layrs*, *lstrp* may be too small), it has been observed to produce convincing results for every problem on which the algorithm has been applied, a few times after adjustments

to one or both parameters. The original ideas for this algorithm are described in [8, 12] in the context of graph bisection and dynamic time warping.

As presented below, *adapt-DP* is essentially an iterative process that restricts its search to the adapting strip around graphs of estimated solutions. Each iteration culminates with the execution of Procedure *DP* for recursively projecting a diffeomorphism obtained from a lower resolution grid to one of higher resolution until full resolution is attained. For simplicity, we assume here $N = M = 2^n + 1$ for some positive integer n. Extending the algorithm to allow N, M to have any values is straightforward. Note, we don't assume partitions $\{t_l\}$, $\{z_l\}$ are uniform. Finally, after the last execution of Procedure *DP* in *adapt-DP*, Procedure *opt-diffeom* is performed to obtain, depending on *layrs* and *lstrp*, optimal γ^* in Γ. Algorithm *adapt-DP* follows:

algorithm *adapt-DP* (DP algorithm)
2. $I(1) = J(1) = 1$
3. $P(N,M) = (1,1)$
for $r = 1$ **to** n **do**
5. $NI = NJ = 2^r + 1$
6. **for** $m = 1$ **to** $NI - 1$ **do**
7. $I(m+1) = m \cdot 2^{n-r} + 1$
8. $r'_m = \frac{1}{2}(t_{I(m)} + t_{I(m+1)})$
end for
for $m = 1$ **to** $NJ - 1$ **do**
$J(m+1) = m \cdot 2^{n-r} + 1$
12. $s'_m = \frac{1}{2}(z_{J(m)} + z_{J(m+1)})$
end for
14. $r'_1 = s'_1 = 0$
15. $r'_{NI-1} = s'_{NJ-1} = 1$
$(i,j) = (N,M)$
$D = \emptyset$
18. **while** $(i,j) \neq (1,1)$ **do**
$(k,l) = P(i,j)$
****
20. Here below, for integers m', n', $1 < m' < NI$,
21. $1 < n' < NJ$, bin $B(m',n') \equiv$
22. $\{(x,y) : r'_{m'-1} \leq x \leq r'_{m'}, s'_{n'-1} \leq y \leq s'_{n'}\}$
****
identify bins $B(m',n')$, $1 < m' < NI$,
$1 < n' < NJ$, the interiors of which are
intersected by line segment $\overline{(i,j)(k,l)}$
$D' = \{(m',n') : \overline{(i,j)(k,l)} \cap B(m',n') \neq \emptyset\}$
$D = D \cup D'$
$(i,j) = (k,l)$

end while
$R = \{(1,1),(N,M)\}$
31. **for** each (m',n') in D **do**
$i_0 = \max\{2, m' - lstrp\}$
$j_0 = \max\{2, n' - lstrp\}$
$R_1 = \{(i,j) : i = I(i'), i_0 \leq i' \leq m', j = J(n')\}$
$R_2 = \{(i,j) : j = J(j'), j_0 \leq j' \leq n', i = I(m')\}$
$R = R \cup R_1 \cup R_2$
end for
38. **execute** procedure *DP* on R
end for
execute procedure *opt-diffeom* to obtain γ^*
end algorithm

In the outline of *adapt-DP* above, we note in line 5, NI starts equal to 3 (for $r = 1$) and then it is essentially doubled at each iteration $r > 1$ until it becomes equal to N at the n^{th} iteration. We note in line 2 and in line 7 inside the **for** loop at line 6, the range of I starts with 3 integers (for $r = 1$) and then essentially doubles in size at each iteration $r > 1$, contains the previous range of I from preceding iteration, and is evenly spread in the set $\{1,2,\ldots,N\}$ until it becomes this set. We note as well from the well-known sum of a geometric series that since $N = 2^n + 1$ then the sum of the NI's, i.e., $(2^1+1)+(2^2+1)+\ldots+(2^n+1)$, is $O(N)$. Clearly, all of the above applies to NJ, M, and the range of J.

We note that the **while** loop at line 18 identifies certain cells in the Voronoi diagram [16] of the set of grid points $R' \equiv \{(i,j) : i = I(m'), j = J(n'), 1 < m' < NI, 1 < n' < NJ\}$ restricted to the unit square. Indeed bin $B(m',n')$ as defined in lines 20-22, in terms of the computations in lines 8, 12, 14, 15, is exactly the Voronoi cell of $(I(m'),J(n'))$, and all such cells together partition the unit square. Accordingly, with γ^* encoded in P in line 3 ($r = 1$) or in line 38 ($r > 1$) through the execution of Procedure *DP* in the previous iteration $(r-1)$, it must be that every point in the graph of γ^* is in some bin $B(m',n')$. Thus, it then seems reasonable to say that a reliable region of influence of γ^* is the region around the graph of γ^* formed by the union of bins within a constant number of bins from the graph. Accordingly, to be precise, a bin B is part of this region if and only if there is a bin B', the interior of which the graph of γ^* intersects, B within a constant number ($lstrp$) of bins from B', B directly below or to the left of B', or B equal to B' (see Figure 2). We note that identifying this region is essentially accomplished in the **while** loop at line 18 and the **for** loop at line 31, with the region understood to be the union of bins or Voronoi cells $B(m',n')$ of grid points in R at the end of the **for** loop. Clearly, the region contains the graph of γ^*, and has the appearance of a strip whose shape evolves from one iteration to the next as it closely mimics the shape of the graph of γ^* (see Figure 2), and thus it is referred to as an adapting strip. Finally, we note that at the end of the **for** loop, γ^* in $\Gamma(R) \subseteq \Gamma(R')$ encoded in P for the current iteration, is obtained in line 38 with Procedure *DP* restricted to the region or adapting strip, a region that as just described depends on all previous γ^* functions from previous iterations. The last γ^* obtained is then, depending on $layrs$, optimal in $\Gamma(R)$,

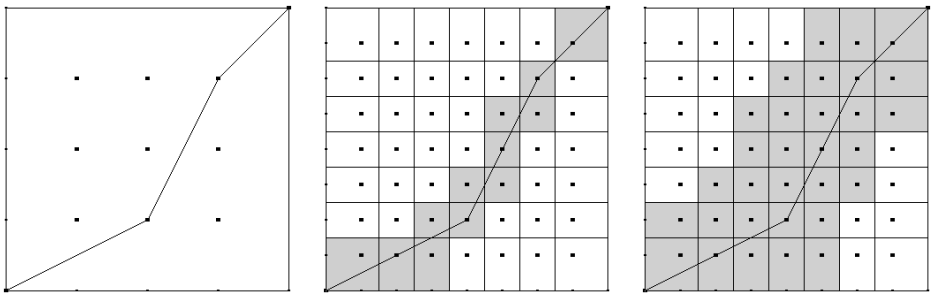

Fig. 2. On left is γ^* from 2^{nd} iteration, $NI = NJ = 2^2 + 1 = 5$. In center, during 3^{rd} iteration, $NI = NJ = 2^3 + 1 = 9$; shaded bins are bins the interior of which γ^* intersects. On right, shaded bins form adapting strip in which next γ^* is computed. Each shaded bin is within 2 bins ($lstrp = 2$) of a bin which is above, to the right of, or equal to it, and whose interior has nonempty intersection with the current γ^*.

and, depending on *layrs* and *lstrp*, in $\Gamma(R')$.

With γ^* as above during the execution of the **while** loop at line 18 for iteration r, we note that since γ^* is in $\Gamma(R)$ then the number of bins $B(m',n')$ whose interiors the graph of γ^* intersects must be $O(NI+NJ)$, which is also the time required to find them one linear component of the graph at a time. Since $|R|$ at end of the **for** loop at line 31 is then $O(lstrp)\cdot O(NI+NJ)$, i.e., $O(NI+NJ)$, the complexity of Procedure *DP* at line 38 is then $O(NI+NJ)$, and since as mentioned above the sum of the NI's and NJ's is $O(N)$ and $O(M)$, respectively, then the complexity of *adapt-DP* must be $O(N+M)$, implying *adapt-DP* is linear.

3. Computation of Rotation for Rigid Alignment of Curves

In this section, we describe and justify in a purely algebraic manner the usual algorithm for computing an optimal rotation for rigid alignment of two simple curves in $d-$dimensional space, d a positive integer, the curves not necessarily closed, each curve with a fixed starting point. Here the computation of the alignment is based only on rotations (with $d\times d$ rotation matrices) of one of the curves. We note, if d equals 1, the only 1×1 rotation matrix possible is the one whose sole entry is 1. For simplicity we say that this matrix equals 1. We also note, results presented here for the justification already appear in [9], although not in the context of shape analysis.

We assume we have two simple curves in $d-$dimensional space represented by functions $\beta_n : [0,1] \to \mathbb{R}^d$, $n = 1,2$, that are absolutely continuous and of unit length, and square-integrable functions $q_n : [0,1] \to \mathbb{R}^d$, $\|q_n\|_{L^2}^2 = \int_0^1 \|q_n(t)\|^2 dt = 1$, $n = 1,2$, the shape functions or SRVF's of β_n, $n = 1,2$, respectively, where $\|\cdot\|$ is the $d-$dimensional Euclidean norm and $\|\cdot\|_{L^2}$ is the L^2 norm for functions in $L^2([0,1],\mathbb{R}^d)$. Given n, $n = 1,2$, we also assume $\beta_n(0)$ is the fixed starting point of β_n. Note, given an absolutely continuous function $\beta : [0,1] \to \mathbb{R}^d$ and R a rotation matrix in $\mathbb{R}^d$, if q is the SRVF of β, then Rq is the SRVF of

$R\beta$ (see [13]). Ideally, an optimal rotation matrix R in $\mathbb{R}^d$ is found, i.e., a $d \times d$ orthogonal matrix R with $\det(R) = 1$ (of determinant 1), that minimizes

$$E(R) = \int_0^1 \|q_1(t) - Rq_2(t)\|^2 dt. \tag{4}$$

For this purpose $E(R)$ in (4) is rewritten as follows:

$$E(R) = \|q_1\|_{L^2}^2 + \|q_2\|_{L^2}^2 - 2\int_0^1 q_1^T(t)Rq_2(t)dt = 2 - 2\int_0^1 q_1^T(t)Rq_2(t)dt$$

(note, $\|Rq_2(t)\| = \|q_2(t)\|$ since R is a rotation matrix). Then minimizing (4) over all rotation matrices R is equivalent to maximizing

$$\int_0^1 q_1^T(t)Rq_2(t)dt = \mathrm{tr}(RA^T),$$

where $\mathrm{tr}(RA^T)$ is trace of RA^T and A is a $d \times d$ matrix with entries $A_{kj} = \int_0^1 q_{1k}(t)q_{2j}(t)dt$, where $q_{1k}(t)$, $q_{2j}(t)$ are the k^{th} and j^{th} coordinates of $q_1(t)$ and $q_2(t)$, respectively, for each pair $k, j = 1, \ldots, d$. We refer to this matrix as the matrix A associated with (4).

As noted in Section 2, in practice, we need to solve a discretized version of the problem. Thus, β_1 and β_2 are given as finite lists of points, say N points per curve for some integer $N > 0$. For some partition $\{t_l\}_{l=1}^N$, $t_1 = 0 < t_2 < \ldots < t_N = 1$, (not necessarily uniform) of $[0,1]$, then for $n = 1,2$, β_n is given as $\beta_n(t_l)$, $l = 1, \ldots, N$. Similarly for q_1, q_2, except that for $l = 1, \ldots, N$, $q_1(t_l)$ and $q_2(t_l)$ are computed as described in Section 2. That the lists for β_1 and β_2, and therefore for q_1 and q_2, must have the same number of points with the same partition is dictated by the way optimal rotation matrices are computed as described below. In what follows, for $l = 1, \ldots, N$, $k = 1, \ldots, d$, $j = 1, \ldots, d$, q_1^l is $q_1(t_l)$, q_2^l is $q_2(t_l)$, q_{1k}^l is the k^{th} coordinate of q_1^l, and q_{2j}^l is the j^{th} coordinate of q_2^l. We then discretize integral (4) using the trapezoidal rule:

$$E^{discr}(R) = \sum_{l=1}^{N-1} 1/2\,(t_{l+1} - t_l)(\|q_1(t_l) - Rq_2(t_l)\|^2 + \|q_1(t_{l+1}) - Rq_2(t_{l+1})\|^2)$$

$$= \sum_{l=1}^{N} h_l \|q_1(t_l) - Rq_2(t_l)\|^2 = \sum_{l=1}^{N} h_l \|q_1^l - Rq_2^l\|^2, \tag{5}$$

where $h_1 = (t_2 - t_1)/2$, $h_N = (t_N - t_{N-1})/2$, and for $l = 2, \ldots, N-1$, $h_l = (t_{l+1} - t_{l-1})/2$. Note, $h_l > 0$ for each l, $l = 1, \ldots, N$, and $\sum_{l=1}^N h_l = 1$.

Note that minimizing (5) over all rotation matrices R is an instance of solving the so-called Wahba's problem [11, 17] which is the problem of minimizing $\sum_{l=1}^N w_l \|q_1^l - Rq_2^l\|^2$ over all rotation matrices R, where the w_l's are nonnegative weights.

Noting $\|Rq_2^l\| = \|q_2^l\|$, $l = 1,\ldots,N$, we can rewrite (5) as follows

$$E^{discr}(R) = \sum_{l=1}^{N} h_l\,(\|q_1^l\|^2 + \|q_2^l\|^2) - 2\sum_{l=1}^{N} h_l\,((q_1^l)^T R q_2^l),$$

so that minimizing (5) over all rotation matrices R is equivalent to maximizing

$$\sum_{l=1}^{N} h_l\,(q_1^l)^T R q_2^l = \mathrm{tr}(RA^T), \tag{6}$$

where A is the $d \times d$ matrix with entries $A_{kj} = \sum_{l=1}^{N} h_l\, q_{1k}^l q_{2j}^l$, for each pair $k, j = 1,\ldots,d$. We refer to this matrix as the matrix A associated with (5). Focusing our attention on this matrix, as opposed to doing it on the matrix A associated with (4), then an optimal rotation matrix R for (6), thus for (5), can be computed from the singular value decomposition of A or, more precisely, with the Kabsch-Umeyama algorithm [6, 7, 15] (see Algorithm Kabsch-Umeyama below, where $\mathrm{diag}\{s_1,\ldots,s_d\}$ is the $d \times d$ diagonal matrix with numbers $s_1,\ldots,s_d$ as the elements of the diagonal, in that order running from the upper left to the lower right of the matrix). A *singular value decomposition* (SVD) [10] of A is a representation of the form $A = USV^T$, where U and V are $d \times d$ orthogonal matrices and S is a $d \times d$ diagonal matrix with the singular values of A, which are nonnegative real numbers, appearing in the diagonal of S in descending order, from the upper left to the lower right of S. Note that any matrix, not necessarily square, has a singular value decomposition, not necessarily unique [10]. Note as well that an optimal rotation matrix R for integral (4) can also be computed in a similar manner using the matrix A associated with (4). However, in what follows, for the obvious reasons, we focus our attention on the discretized integral (5). Finally, note that applications of the Kabsch-Umeyama algorithm similar to the one just decribed here can be found in [5, 13].

Algorithm Kabsch-Umeyama (KU algorithm)

Set $h_1 = (t_2 - t_1)/2$, $h_N = (t_N - t_{N-1})/2$, $h_l = (t_{l+1} - t_{l-1})/2$ for $l = 2,\ldots,N-1$.
Set q_{1k}^l equal to k^{th} coordinate of $q_1(t_l)$ for $l = 1,\ldots,N$, $k = 1,\ldots,d$.
Set q_{2j}^l equal to j^{th} coordinate of $q_2(t_l)$ for $l = 1,\ldots,N$, $j = 1,\ldots,d$.
Compute $A_{kj} = \sum_{l=1}^{N} h_l\, q_{1k}^l q_{2j}^l$ for each pair $k, j = 1,\ldots,d$.
Identify $d \times d$ matrix A with entries A_{kj} for each pair $k, j = 1,\ldots,d$.
Compute SVD of A, i.e., identify $d \times d$ matrices U, S, V, so that $A = USV^T$
Set $s_1 = \ldots = s_{d-1} = 1$.
if $\det(UV) > 0$ **then** set $s_d = 1$.
else set $s_d = -1$. **end if**
Set $\tilde{S} = \mathrm{diag}\{s_1,\ldots,s_d\}$.
Compute and return $R = U\tilde{S}V^T$ and $maxtrace = \mathrm{tr}(RA^T)$.

We note that if $d = 1$, the KU algorithm still computes R and *maxtrace*, with the resulting R always equal to 1. We should also note that alternative methods for $d = 2,\ 3$ to the Kabsch-Umeyama algorithm, i.e., that do not use the SVD to compute a rotation matrix R that maximizes $\text{tr}(RM)$ over all $d \times d$ rotations matrices, M a $d \times d$ matrix, have been presented in [3].

Now we justify the KU algorithm using exclusively simple concepts from linear algebra, mostly in the proof of the proposition that follows. The algorithm has been previously justified in [6, 7, 15], however the justifications there are not totally algebraic as they are based on the optimization technique of Langrange multipliers. The justification here already appears in [9], however it was not developed there in the context of shape analysis. Accordingly we develop it here with that context in mind (the matrix A in the outline of the KU algorithm above is defined in that context) but mainly for self-containment and clarity.

Proposition 1: If $D = \text{diag}\{\sigma_1, \ldots, \sigma_d\}$, $\sigma_j \geq 0$, $j = 1, \ldots, d$, and W is a $d \times d$ orthogonal matrix, then
1. $\text{tr}(WD) \leq \sum_{j=1}^{d} \sigma_j$.
2. If B is a $d \times d$ orthogonal matrix, $S = B^T DB$, then $\text{tr}(WS) \leq \text{tr}(S)$.
3. If $\det(W) = -1$, $\sigma_d \leq \sigma_j$, $j = 1, \ldots, d-1$, then $\text{tr}(WD) \leq \sum_{j=1}^{d-1} \sigma_j - \sigma_d$.

Proof: Since W is orthogonal and if W_{kj}, $k, j = 1, \ldots, d$, are the entries of W, then, in particular, $W_{jj} \leq 1$, $j = 1, \ldots, d$, so that
$\text{tr}(WD) = \sum_{j=1}^{d} W_{jj}\sigma_j \leq \sum_{j=1}^{d} \sigma_j$, and therefore 1. holds.
Accordingly, assumming B is a $d \times d$ orthogonal matrix, since BWB^T is also orthogonal, it follows from 1. that
$\text{tr}(WS) = \text{tr}(WB^T DB) = \text{tr}(BWB^T D) \leq \sum_{j=1}^{d} \sigma_j = \text{tr}(D) = \text{tr}(S)$, and therefore 2. holds.
If $\det(W) = -1$, we show next that a $d \times d$ orthogonal matrix B can be identified so that with $\bar{W} = B^T WB$, then $\bar{W} = \begin{pmatrix} W_0 & O \\ O^T & -1 \end{pmatrix}$, W_0 interpreted as the upper leftmost $d-1 \times d-1$ entries of $\bar{W}$ and as a $d-1 \times d-1$ matrix as well; O interpreted as a vertical column or vector of $d-1$ zeroes.
With I as the $d \times d$ identity matrix, then $\det(W) = -1$ implies $\det(W+I) = -\det(W)\det(W+I) = -\det(W^T)\det(W+I) = -\det(I+W^T) = -\det(I+W)$ which implies $\det(W+I) = 0$ so that $x \neq 0$ exists in $\mathbb{R}^d$ with $Wx = -x$. It also follows then that $W^T Wx = W^T(-x)$ which gives $x = -W^T x$ so that $W^T x = -x$ as well.
Letting $b_d = x$, vectors $b_1, \ldots, b_{d-1}$ can be obtained so that $b_1, \ldots, b_d$ form a basis of $\mathbb{R}^d$, and by the Gram-Schmidt process starting with b_d, we may assume $b_1, \ldots, b_d$ form an orthonormal basis of $\mathbb{R}^d$ with $Wb_d = W^T b_d = -b_d$. Letting $B = (b_1, \ldots, b_d)$, interpreted as a $d \times d$ matrix with columns $b_1, \ldots, b_d$, in that order, then it follows that B is orthogonal, and with $\bar{W} = B^T WB$ and W_0, O as previously described, noting $B^T Wb_d = B^T(-b_d) = \begin{pmatrix} O \\ -1 \end{pmatrix}$ and $b_d^T WB = (W^T b_d)^T B = (-b_d)^T B = (O^T\ -1)$, then $\bar{W} = \begin{pmatrix} W_0 & O \\ O^T & -1 \end{pmatrix}$. Note, $\bar{W}$ is orthogonal and therefore so is the $d-1 \times d-1$ matrix W_0.

Let $S = B^T DB$ and write $S = \begin{pmatrix} S_0 & a \\ b^T & \gamma \end{pmatrix}$, S_0 interpreted as the upper leftmost $d-1 \times d-1$ entries of S and as a $d-1 \times d-1$ matrix as well; a and b interpreted as vertical columns or vectors of $d-1$ entries, and γ as a scalar.
Note, $\mathrm{tr}(WD) = \mathrm{tr}(B^T WDB) = \mathrm{tr}(B^T WBB^T DB) = \mathrm{tr}(\bar{W}S)$, so that $\bar{W}S = \begin{pmatrix} W_0 & O \\ O^T & -1 \end{pmatrix} \begin{pmatrix} S_0 & a \\ b^T & \gamma \end{pmatrix} = \begin{pmatrix} W_0 S_0 & W_0 a \\ -b^T & -\gamma \end{pmatrix}$ gives $\mathrm{tr}(WD) = \mathrm{tr}(W_0 S_0) - \gamma$.
We show $\mathrm{tr}(W_0 S_0) \leq \mathrm{tr}(S_0)$. For this purpose let $\hat{W} = \begin{pmatrix} W_0 & O \\ O^T & 1 \end{pmatrix}$, W_0 and O as above. Since W_0 is orthogonal, then clearly $\hat{W}$ is a $d \times d$ orthogonal matrix, and by 2. $\mathrm{tr}(\hat{W}S) \leq \mathrm{tr}(S)$ so that $\hat{W}S = \begin{pmatrix} W_0 & O \\ O^T & 1 \end{pmatrix} \begin{pmatrix} S_0 & a \\ b^T & \gamma \end{pmatrix} = \begin{pmatrix} W_0 S_0 & W_0 a \\ b^T & \gamma \end{pmatrix}$ gives $\mathrm{tr}(W_0 S_0) + \gamma = \mathrm{tr}(\hat{W}S) \leq \mathrm{tr}(S) = \mathrm{tr}(S_0) + \gamma$.
Thus, $\mathrm{tr}(W_0 S_0) \leq \mathrm{tr}(S_0)$.
Note, $\mathrm{tr}(S_0) + \gamma = \mathrm{tr}(S) = \mathrm{tr}(D)$, and if B_{kj}, $k, j = 1, \ldots, d$ are the entries of B, then $\gamma = \sum_{k=1}^{d} B_{kd}^2 \sigma_k$, a convex combination of the σ_k's, so that $\gamma \geq \sigma_d$. It then follows that $\mathrm{tr}(WD) = \mathrm{tr}(W_0 S_0) - \gamma \leq \mathrm{tr}(S_0) - \gamma = \mathrm{tr}(D) - \gamma - \gamma \leq \sum_{j=1}^{d-1} \sigma_j - \sigma_d$, and therefore 3. holds. □

Finally, the following theorem, which is a consequence of Proposition 1, justifies the Kabsch-Umeyama algorithm.

Theorem: Given a $d \times d$ matrix M, let U, S, V be $d \times d$ matrices such that the singular value decomposition of M gives $M = USV^T$. If $\det(UV^T) > 0$, then $R = UV^T$ maximizes $\mathrm{tr}(RM^T)$ over all $d \times d$ rotation matrices R. Otherwise, if $\det(UV^T) < 0$, with $\tilde{S} = \mathrm{diag}\{s_1, \ldots, s_d\}$, $s_1 = \ldots = s_{d-1} = 1$, $s_d = -1$, then $R = U\tilde{S}V^T$ maximizes $\mathrm{tr}(RM^T)$ over all $d \times d$ rotation matrices R.

Proof: Let σ_j, $j = 1, \ldots, d$, $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_d \geq 0$, be the singular values of M, so that $S = \mathrm{diag}\{\sigma_1, \ldots, \sigma_d\}$.
Assume $\det(UV^T) > 0$. If R is any rotation matrix, then R is orthogonal. From 1. of Proposition 1 since $U^T RV$ is orthogonal, then
$\mathrm{tr}(RM^T) = \mathrm{tr}(RVSU^T) = \mathrm{tr}(U^T RVS) \leq \sum_{j=1}^{d} \sigma_j$.
On the other hand, if $R = UV^T$, then R is clearly orthogonal, $\det(R) = 1$, and $\mathrm{tr}(RM^T) = \mathrm{tr}(UV^T VSU^T) = \mathrm{tr}(USU^T) = \mathrm{tr}(S) = \sum_{j=1}^{d} \sigma_j$.
Thus, $R = UV^T$ maximizes $\mathrm{tr}(RM^T)$ over all $d \times d$ rotation matrices R.
Finally, assume $\det(UV^T) < 0$. If R is any rotation matrix, then R is orthogonal and $\det(R) = 1$. From 3. of Proposition 1 since $U^T RV$ is orthogonal and $\det(U^T RV) = -1$, then
$\mathrm{tr}(RM^T) = \mathrm{tr}(RVSU^T) = \mathrm{tr}(U^T RVS) \leq \sum_{j=1}^{d-1} \sigma_j - \sigma_d$.
On the other hand, if $R = U\tilde{S}V^T$, then R is clearly orthogonal, $\det(R) = 1$, and $\mathrm{tr}(RM^T) = \mathrm{tr}(U\tilde{S}V^T VSU^T) = \mathrm{tr}(U\tilde{S}SU^T) = \mathrm{tr}(\tilde{S}S) = \sum_{j=1}^{d-1} \sigma_j - \sigma_d$.
Thus, $R = U\tilde{S}V^T$ maximizes $\mathrm{tr}(RM^T)$ over all $d \times d$ rotation matrices R. □

In the rest of this section, although not exactly related to the goal of this paper, for the sake of completeness, we show how another problem of interest reduces to the problem just solved above so that it can then be solved with the Kabsch-Umeyama algorithm. The

problem of interest is the so-called orientation-preserving rigid motion problem. With q_1, q_2, h_l, q_1^l, q_2^l, $l = 1,\ldots,N$, as above, the problem is then that of finding an orientation-preserving rigid motion ϕ of $\mathbb{R}^d$ that minimizes

$$\Delta(\phi) = \sum_{l=1}^{N} h_l \, \|q_1^l - \phi(q_2^l)\|^2. \tag{7}$$

An affine linear function ϕ, $\phi : \mathbb{R}^d \to \mathbb{R}^d$, is a rigid motion of $\mathbb{R}^d$ if it is of the form $\phi(q) = Rq + t$ for $q \in \mathbb{R}^d$, where R is a $d \times d$ orthogonal matrix, and $t \in \mathbb{R}^d$. The rigid motion ϕ is orientation preserving if $\det(R) = 1$. We note that the justification of the reduction here, apparently already appears in [9]. However there the problem being reduced does not involve a partition of $[0,1]$ the way it does here, a partition that can be either uniform or nonuniform, although, if so desired, it is not hard to show that a partition can actually be associated with the problem in [9], a partition that must be uniform. Thus the problem being reduced here is more general than the problem in [9].

Let $\bar{q}_1$ and $\bar{q}_2$ denote the centroids of the discretized q_1 and q_2, respectively:

$$\bar{q}_1 = \sum_{l=1}^{N} h_l \, q_1^l \quad \text{and} \quad \bar{q}_2 = \sum_{l=1}^{N} h_l \, q_2^l.$$

The following proposition shows, in particular, that $\phi(\bar{q}_2) = \bar{q}_1$ if ϕ minimizes (7) over either the set of all rigid motions of $\mathbb{R}^d$ or the smaller set of rigid motions of $\mathbb{R}^d$ that are orientation preserving.

Proposition 2: Let ϕ be a rigid motion of $\mathbb{R}^d$ with $\phi(\bar{q}_2) \neq \bar{q}_1$ and define an affine linear function τ, $\tau : \mathbb{R}^d \to \mathbb{R}^d$, $\tau(q) = \phi(q) - \phi(\bar{q}_2) + \bar{q}_1$ for $q \in \mathbb{R}^d$. Then τ is a rigid motion of $\mathbb{R}^d$, $\tau(\bar{q}_2) = \bar{q}_1$, $\Delta(\tau) < \Delta(\phi)$, and if ϕ is orientation preserving, then so is τ.

Proof: Clearly $\tau(\bar{q}_2) = \bar{q}_1$. Let R be a $d \times d$ orthogonal matrix and $t \in \mathbb{R}^d$ such that $\phi(q) = Rq + t$ for q in $\mathbb{R}^d$. Then $\tau(q) = Rq - R\bar{q}_2 + \bar{q}_1$ so that τ is a rigid motion of $\mathbb{R}^d$, τ is orientation preserving if ϕ is, and for each l, $l = 1,\ldots,N$, we have

$$\begin{aligned}
&\|q_1^l - \phi(q_2^l)\|^2 - \|q_1^l - \tau(q_2^l)\|^2 \\
&= (q_1^l - Rq_2^l - t)^T (q_1^l - Rq_2^l - t) \\
&\quad -(q_1^l - Rq_2^l + R\bar{q}_2 - \bar{q}_1)^T (q_1^l - Rq_2^l + R\bar{q}_2 - \bar{q}_1) \\
&= (q_1^l - Rq_2^l)^T (q_1^l - Rq_2^l) - 2(q_1^l - Rq_2^l)^T t + t^T t - (q_1^l - Rq_2^l)^T (q_1^l - Rq_2^l) \\
&\quad -2(q_1^l - Rq_2^l)^T (R\bar{q}_2 - \bar{q}_1) - (R\bar{q}_2 - \bar{q}_1)^T (R\bar{q}_2 - \bar{q}_1) \\
&= -2(q_1^l - Rq_2^l)^T t + t^T t - 2(q_1^l - Rq_2^l)^T (R\bar{q}_2 - \bar{q}_1) \\
&\quad -(R\bar{q}_2 - \bar{q}_1)^T (R\bar{q}_2 - \bar{q}_1) + 2(R\bar{q}_2 - \bar{q}_1)^T t - 2(R\bar{q}_2 - \bar{q}_1)^T t + t^T t - t^T t \\
&= 2(Rq_2^l - q_1^l)^T t + 2t^T t + 2(Rq_2^l - q_1^l)^T (R\bar{q}_2 - \bar{q}_1) + 2(R\bar{q}_2 - \bar{q}_1)^T t \\
&\quad -((R\bar{q}_2 - \bar{q}_1)^T (R\bar{q}_2 - \bar{q}_1) + 2(R\bar{q}_2 - \bar{q}_1)^T t + t^T t) \\
&= 2(Rq_2^l - q_1^l + t)^T (R\bar{q}_2 - \bar{q}_1 + t) - (R\bar{q}_2 - \bar{q}_1 + t)^T (R\bar{q}_2 - \bar{q}_1 + t).
\end{aligned}$$

It then follows that

$\Delta(\phi)-\Delta(\tau)=\sum_{l=1}^N h_l\|q_1^l-\phi(q_2^l)\|^2-\sum_{l=1}^N h_l\|q_1^l-\tau(q_2^l)\|^2$
$=\sum_{l=1}^N h_l(\|q_1^l-\phi(q_2^l)\|^2-\|q_1^l-\tau(q_2^l)\|^2)$
$=\sum_{l=1}^N h_l(2(Rq_2^l-q_1^l+t)^T(R\bar{q}_2-\bar{q}_1+t)-(R\bar{q}_2-\bar{q}_1+t)^T(R\bar{q}_2-\bar{q}_1+t))$
$=2(R\sum_{l=1}^N h_l q_2^l-\sum_{l=1}^N h_l q_1^l+t\sum_{l=1}^N h_l)^T(R\bar{q}_2-\bar{q}_1+t)$
$\quad-(R\bar{q}_2-\bar{q}_1+t)^T(R\bar{q}_2-\bar{q}_1+t)\sum_{l=1}^N h_l$
$=2(R\bar{q}_2-\bar{q}_1+t)^T(R\bar{q}_2-\bar{q}_1+t)-(R\bar{q}_2-\bar{q}_1+t)^T(R\bar{q}_2-\bar{q}_1+t)$
$=(R\bar{q}_2-\bar{q}_1+t)^T(R\bar{q}_2-\bar{q}_1+t)=\|R\bar{q}_2-\bar{q}_1+t\|^2=\|\phi(\bar{q}_2)-\bar{q}_1\|^2>0.$ □

Finally, the following corollary is a consequence of Proposition 2. Here $p_1^l=q_1^l-\bar{q}_1$, $p_2^l=q_2^l-\bar{q}_2$, for $l=1,\ldots,N$, and if $\bar{p}_1=\sum_{l=1}^N h_l p_1^l$, $\bar{p}_2=\sum_{l=1}^N h_l p_2^l$, then clearly $\bar{p}_1=\bar{p}_2=0$. It shows that the problem of finding an orientation-preserving rigid motion ϕ of $\mathbb{R}^d$ that minimizes (7) can be reduced to the problem of finding a $d\times d$ rotation matrix R that minimizes $\sum_{l=1}^N h_l\|p_1^l-Rp_2^l\|^2$ which, of course, then can be solved with the Kabsch-Umeyama algorithm.

Corollary: Let $\hat{R}$ be such that $R=\hat{R}$ minimizes $\sum_{l=1}^N h_l\|p_1^l-Rp_2^l\|^2$ over all $d\times d$ rotation matrices R. Let $\hat{t}=\bar{q}_1-\hat{R}\bar{q}_2$, and let $\hat{\phi}$ be given by $\hat{\phi}(q)=\hat{R}q+\hat{t}$ for $q\in\mathbb{R}^d$. Then $\phi=\hat{\phi}$ minimizes $\sum_{l=1}^N h_l\|q_1^l-\phi(q_2^l)\|^2$ over all orientation-preserving rigid motions ϕ of $\mathbb{R}^d$.

Proof: One such $\hat{R}$ can be computed with the Kabsch-Umeyama algorithm. By Proposition 2, in order to minimize $\sum_{l=1}^N h_l\|q_1^l-\phi(q_2^l)\|^2$ over all orientation-preserving rigid motions ϕ of $\mathbb{R}^d$, it suffices to do it over those for which $\phi(\bar{q}_2)=\bar{q}_1$. Therefore, it suffices to minimize $\sum_{l=1}^N h_l\|q_1^l-(Rq_2^l+t)\|^2$ with $t=\bar{q}_1-R\bar{q}_2$ over all $d\times d$ rotation matrices R, i.e., it suffices to minimize

$$\sum_{l=1}^N h_l\|q_1^l-Rq_2^l-\bar{q}_1+R\bar{q}_2\|^2=\sum_{l=1}^N h_l\|(q_1^l-\bar{q}_1)-R(q_2^l-\bar{q}_2)\|^2$$

over all $d\times d$ rotation matrices R. But minimizing the last expression is equivalent to minimizing $\sum_{l=1}^N h_l\|p_1^l-Rp_2^l\|^2$ over all $d\times d$ rotation matrices R. Since $R=\hat{R}$ is a solution to this last problem, it then follows that $R=\hat{R}$ minimizes $\sum_{l=1}^N h_l\|q_1^l-Rq_2^l-\bar{q}_1+R\bar{q}_2\|^2=\sum_{l=1}^N h_l\|q_1^l-(Rq_2^l+t)\|^2$ with $t=\bar{q}_1-R\bar{q}_2$ over all $d\times d$ rotation matrices R. Consequently, if $\hat{t}=\bar{q}_1-\hat{R}\bar{q}_2$, and $\hat{\phi}$ is given by $\hat{\phi}(q)=\hat{R}q+\hat{t}$ for $q\in\mathbb{R}^d$, then $\phi=\hat{\phi}$ clearly minimizes $\sum_{l=1}^N h_l\|q_1^l-\phi(q_2^l)\|^2$ over all orientation-preserving rigid motions ϕ of $\mathbb{R}^d$. □

4. Computation of the Elastic Shape Distance between Two Curves

Let β_1, β_2, q_1, q_2 be as above, i.e., $\beta_n:[0,1]\to\mathbb{R}^d$, $n=1,2$, are absolutely continuous functions representing simple curves in $\mathbb{R}^d$ of unit length, and $q_n:[0,1]\to\mathbb{R}^d$, $n=1,2$, are square-integrable functions that are the shape functions or SRVF's of β_n, $n=1,2$, respectively. In the case one of the curves is closed, say β_1, then, in particular, q_1 is interpreted to be a periodic function from $\mathbb{R}$ into $\mathbb{R}^d$ so that $q_1(t+1)=q_1(t)$ for all vaues of t. Define a finite subset K of $[0,1]$ as follows. If neither curve is closed let $K=\{0\}$. Otherwise, assume the curve represented by β_1, the first curve, is closed, and for an integer $T>0$ choose

numbers $\hat{t}_l$, $l = 1,\ldots,T$, in $[0,1]$, $\hat{t}_1 = 0 < \ldots < \hat{t}_T = 1$, so that $B = \{\beta_1(\hat{t}_l), l = 1,\ldots,T\}$ is reasonably dense in β_1 (by joining consecutive points in B with line segments, the resulting piecewise linear curve should be a reasonable approximation of β_1). Accordingly, let $K = \{\hat{t}_1,\ldots,\hat{t}_T\}$ in this case. Either way, K is a finite subset of $[0,1]$ and $\{\beta_1(t), t \in K\}$ is interpreted to be the set of starting points of β_1. With $SO(d)$ as the set of $d \times d$ rotation matrices, and Γ as the set of diffeomorphisms of $[0,1]$ onto itself ($\gamma(0) = 0$, $\gamma(1) = 1$, $\dot{\gamma} > 0$, for $\gamma \in \Gamma$), given $t_0 \in K$, $R \in SO(d)$, $\gamma \in \Gamma$, a mismatch energy $E(t_0, R, \gamma)$ is defined by

$$E(t_0, R, \gamma) = \int_0^1 \| \sqrt{\dot{\gamma}(t)} R q_1(t_0 + \gamma(t)) - q_2(t) \|^2 dt \tag{8}$$

which as noted in [5], for the purpose of minimizing it with respect to t_0, R and γ, without any loss of generality, can be reformulated as

$$E(t_0, R, \gamma) = \int_0^1 \| R q_1(t_0 + t) - \sqrt{\dot{\gamma}(t)} q_2(\gamma(t)) \|^2 dt \tag{9}$$

which allows for the second curve to be reparametrized while the first one is rotated. Note, $SO(d) = \{1\}$ if $d = 1$.

As established in [13, 14], given $t_0 \in K$, $R \in SO(d)$, $\gamma \in \Gamma$, so that the triple (t_0, R, γ) is a global minimizer of (8), then $E(t_0, R, \gamma)$ can be interpreted to be the elastic shape distance between β_1 and β_2, and the elastic registration of β_1 and β_2 is obtained by reparametrizing and rotating β_1 with γ and R, respectively, with starting point $\beta_1(t_0)$. Note, if β_1 is closed, we assume $\dot{\gamma}(0) = \dot{\gamma}(1)$ to ensure the periodicity of $\sqrt{\dot{\gamma}(t)} R q_1(t_0 + \gamma(t))$ for $t \in [0,1]$. Similarly, if (t_0, R, γ) is a global minimizer of (9) then again $E(t_0, R, \gamma)$ is interpreted to be the elastic shape distance between β_1 and β_2, and the elastic registration of β_1 and β_2 is obtained by rotating β_1 with R, with starting point $\beta_1(t_0)$, and reparametrizing β_2 with γ.

As noted in Section 2 and Section 3, in practice, we work with curves β_1, β_2, given as finite lists of points in the curves. Unless otherwise specified, here and in what follows, for simplicity, although mostly out of necessity as will be made clear below, we assume that β_1 and β_2 are discretized by the same partition of $[0,1]$. Even if at first they are not, this can be easily accomplished by interpolating with a cubic spline one or both curves. Accordingly, for some integer $N > 0$, and a partition of $[0,1]$, $\{t_l\}_{l=1}^N$, $t_1 = 0 < t_2 < \ldots < t_N = 1$, for $n = 1,2$, the curve β_n is given as a list of N points in the curve, where for $l = 1,\ldots,N$, $\beta_n(t_l)$ is the l^{th} point in the list for β_n. Similarly for q_1, q_2, except that for $l = 1,\ldots,N$, $q_1(t_l)$ and $q_2(t_l)$ are computed as described in Section 2. That the lists for β_1 and β_2, and therefore for q_1 and q_2, must have the same number of points with the same partition is dictated by the way optimal rotation matrices are computed as described in Section 3. If neither curve is closed so that each curve has a fixed starting point, then the partition does not have to be uniform as pointed out in Section 3. The starting points of β_1 and β_2 are then $\beta_1(t_1) = \beta_1(0)$ and $\beta_2(t_1) = \beta_2(0)$, respectively, and as pointed out above the finite subset K of $[0,1]$ defined above must equal $\{0\} = \{t_1\}$. However if one curve is closed, assumed to be β_1, then β_2 has a fixed starting point which is $\beta_2(0)$, and any point in β_1 can then be treated as a starting point of β_1. In this case, for simplicity, K is purposely chosen

to equal $\{t_1,\ldots,t_{N-1}\}$ or a subset of it, so that $B=\{\beta_1(t),t\in K\}$ can be assumed to be reasonably dense in β_1 and therefore essentially of size $O(N)$. B is then interpreted to be the set of starting points of the curve, and if the partition $\{t_l\}_{l=1}^N$ is uniform, an optimal rotation matrix can then be easily computed as described in Section 3 for each starting point in B. We note that if K equals $\{t_1,\ldots,t_{N-1}\}$, then the partition must indeed be uniform. To see this we note that, in particular, for each m, $m=1,\ldots,N-1$, it should be that $t_m+t_2=t_{m+1}$ so that $t_{m+1}-t_m=t_2=t_2-t_1$ and therefore the partition is uniform. Even if K is not all of $\{t_1,\ldots,t_{N-1}\}$ but a subset of it so that $B=\{\beta_1(t),t\in K\}$ is reasonably dense in β_1, again, for simplicity, we still assume the partition is uniform. Thus, with K as above, for any d, given $t_0\in K$, $R\in SO(d)$, $\gamma\in\Gamma$, we discretize (8) with the trapezoidal rule:

$$E^{discr}(t_0,R,\vec{\gamma})=\sum_{l=1}^{N}h'_l\,\|\sqrt{\dot{\gamma}_l}Rq_1(t_0+\gamma_l)-q_2(t_l)\|^2, \tag{10}$$

where $h'_1=(t_2-t_1)/2$, $h'_N=(t_N-t_{N-1})/2$, $h'_l=(t_{l+1}-t_{l-1})/2$ for $l=2,\ldots,N-1$, $\vec{\gamma}=(\gamma_l)_{l=1}^N$, $\gamma_1=0,\gamma_N=1$, $\gamma_l=\gamma(t_l)$, $\dot{\gamma}_l=(\gamma_{l+1}-\gamma_l)/h_l$ for $l=1,\ldots,N-1$, $\dot{\gamma}_N=\dot{\gamma}_1$, $h_l=t_{l+1}-t_l$ for $l=1,\ldots,N-1$, and $q_1(t_0+\gamma_l)$, $l=1,\ldots,N$, are approximations of q_1 at each value of $t_0+\gamma_l$ obtained from the interpolation of $q_1(t_l)$, $l=1,\ldots,N$, by a cubic spline. Accordingly, in [13, 14] (10) is minimized using a procedure based on alternating computations of approximately optimal diffeomorphisms (for reparametrizing the first curve) and approximately optimal rotation matrices (for rotating the first curve) computed as described in Section 2 and Section 3, respectively, one starting point of β_1 at a time. For the sake of completeness, the procedure, with d, K, $q_1(t_l)$, $q_2(t_l)$, $l=1,\ldots,N$, as input, is summarized in Procedure 1 below. We note that in Section 2 and Section 3 computations are carried out that involve shape functions q_1, q_2, that if discretized by the same partition $\{t_l\}_{l=1}^N$ are given as finite lists of points: $q_1(t_l)$, $q_2(t_l)$, $l=1,\ldots,N$. Thus, in what follows, given shape functions q, $\hat{q}$, to say "Execute DP algorithm for $q(t_l)$, $\hat{q}(t_l)$, $l=1,\ldots,N$" will mean the DP algorithm (*adapt-DP*) should be executed with q, $\hat{q}$ taking the place of q_1, q_2, respectively, in the DP algorithm as outlined in Section 2. The same for the Kabsch-Umeyama algorithm, i.e., the KU algorithm, as outlined in Section 3, and in the next section the KU2 algorithm which is the Kabsch-Umeyama algorithm using the FFT.

Procedure 1

Set $h_l=t_{l+1}-t_l$ for $l=1,\ldots,N-1$.
Set $h'_1=(t_2-t_1)/2$, $h'_N=(t_N-t_{N-1})/2$, $h'_l=(t_{l+1}-t_{l-1})/2$ for
$l=2,\ldots,N-1$.
for each $t_0\in K$ **do**
  Set $\hat{q}_1(t_l)=q_1(t_0+t_l)$ for $l=1,\ldots,N$.
  if $d\neq 1$ **then** execute KU algorithm for $q_2(t_l)$, $\hat{q}_1(t_l)$, $l=1,\ldots,N$,
  to get rotation matrix R.
  else set $R=1$. **end if**
  Set $\bar{q}_1(t_l)=R\hat{q}_1(t_l)$ for $l=1,\ldots,N$.

Execute DP algorithm for $q_2(t_l)$, $\bar{q}_1(t_l)$, $l = 1,\ldots,N$, to get
discretized diffeomorphism $\vec{\gamma} = (\gamma_l)_{l=1}^N$.
Set $\dot{\gamma}_l = (\gamma_{l+1} - \gamma_l)/h_l$ for $l = 1,\ldots,N-1$, $\dot{\gamma}_N = \dot{\gamma}_1$.
From interpolation of $q_1(t_l)$, $l = 1,\ldots,N$, with a cubic spline
set $\hat{q}_1(t_l) = \sqrt{\dot{\gamma}_l} q_1(t_0 + \gamma_l)$ for $l = 1,\ldots,N$.
if $d \neq 1$ **then** execute KU algorithm for $q_2(t_l)$, $\hat{q}_1(t_l)$, $l = 1,\ldots,N$,
to get rotation matrix R.
else set $R = 1$. **end if**
Set $\hat{q}_1(t_l) = R\hat{q}_1(t_l)$ for $l = 1,\ldots,N$.
Compute $E^{discr}(t_0, R, \vec{\gamma}) = \sum_{l=1}^N h'_l \, \|\sqrt{\dot{\gamma}_l} R q_1(t_0 + \gamma_l) - q_2(t_l)\|^2$
$= \sum_{l=1}^N h'_l \, \|\hat{q}_1(t_l) - q_2(t_l)\|^2$.
Keep track of triple $(t_0, R, \vec{\gamma})$, $E^{discr}(t_0, R, \vec{\gamma})$,
$\hat{q}_1(t_l)$, $l = 1,\ldots,N$, with smallest value for $E^{discr}(t_0, R, \vec{\gamma})$.
end for
From interpolation of $\beta_1(t_l)$, $l = 1,\ldots,N$, with a cubic spline
set $\hat{\beta}_1(t_l) = R\beta_1(t_0 + \gamma_l)$ for $l = 1,\ldots,N$.
Return $(t_0, R, \vec{\gamma})$, $E^{discr}(t_0, R, \vec{\gamma})$, $\hat{\beta}_1(t_l)$, $\hat{q}_1(t_l)$, $l = 1,\ldots,N$.

On output, $E^{discr}(t_0, R, \vec{\gamma})$ is interpreted to be the elastic shape distance between β_1 and β_2. On the other hand, $\hat{\beta}_1(t_l)$ and $\beta_2(t_l)$, $l = 1,\ldots,N$, are interpreted to achieve the elastic registration of β_1 and β_2.

Although Procedure 1 above is set up to handle the particular case in which d equals 1 and neither curve is closed, we note, for the obvious reasons, that if Procedure 1 is adjusted appropriately, the requirement that the two curves β_1 and β_2 must have the same number of points with the same partition, is then no longer necessary for this case, this case then being the only case in which the requirement can be ignored. The adjusted Procedure 1, with $q_1(t_l)$, $l = 1,\ldots,N$, $q_2(z_j)$, $j = 1,\ldots,M$, as input, is summarized in Procedure 1' below. There to say "Execute DP algorithm for $q_1(t_l)$, $l = 1,\ldots,N$, $q_2(z_j)$, $j = 1,\ldots,M$" will mean the DP algorithm (*adapt-DP*) should be executed with q_1, q_2 exactly as q_1, q_2 appear in the DP algorithm as outlined in Section 2.

Procedure 1'

Set $h_l = t_{l+1} - t_l$ for $l = 1,\ldots,N-1$.
Set $h'_1 = (t_2 - t_1)/2$, $h'_N = (t_N - t_{N-1})/2$, $h'_l = (t_{l+1} - t_{l-1})/2$ for
$l = 2,\ldots,N-1$.
Execute DP algorithm for $q_1(t_l)$, $l = 1,\ldots,N$, $q_2(z_j)$, $j = 1,\ldots,M$,
to get discretized diffeomorphism $\vec{\gamma} = (\gamma_l)_{l=1}^N$.
Set $\dot{\gamma}_l = (\gamma_{l+1} - \gamma_l)/h_l$ for $l = 1,\ldots,N-1$, $\dot{\gamma}_N = \dot{\gamma}_1$.
From interpolation of $q_2(z_j)$, $j = 1,\ldots,M$, with a cubic spline
set $\hat{q}_2(t_l) = \sqrt{\dot{\gamma}_l} q_2(\gamma_l)$ for $l = 1,\ldots,N$.

Compute $E^{discr}(0,1,\vec{\gamma}) = \sum_{l=1}^{N} h'_l \, \|q_1(t_l) - \sqrt{\dot{\gamma}_l} q_2(\gamma_l)\|^2$
$= \sum_{l=1}^{N} h'_l \, \|q_1(t_l) - \hat{q}_2(t_l)\|^2$.
From interpolation of $\beta_2(z_j)$, $j = 1,\ldots,M$, with a cubic spline
set $\hat{\beta}_2(t_l) = \beta_2(\gamma_l)$ for $l = 1,\ldots,N$.
Return $(0,1,\vec{\gamma})$, $E^{discr}(0,1,\vec{\gamma})$, $\hat{\beta}_2(t_l)$, $\hat{q}_2(t_l)$, $l = 1,\ldots,N$.

On output, $E^{discr}(0,1,\vec{\gamma})$ is interpreted to be the elastic shape distance between β_1 and β_2. On the other hand, $\beta_1(t_l)$ and $\hat{\beta}_2(t_l)$, $l = 1,\ldots,N$, are interpreted to achieve the elastic registration of β_1 and β_2.

Finally, with K, $q_1(t_l)$, $q_2(t_l)$, $l = 1,\ldots,N$, as above, and the partition $\{t_l\}_{l=1}^{N}$ uniform if at least one of the curves is closed, given $t_0 \in K$, $R \in SO(d)$, $\gamma \in \Gamma$, we discretize (9) with the trapezoidal rule as follows:

$$E^{discr}(t_0,R,\vec{\gamma}) = \sum_{l=1}^{N} h'_l \, \|Rq_1(t_0+t_l) - \sqrt{\dot{\gamma}_l} Rq_2(\gamma_l)\|^2, \tag{11}$$

where $h'_1 = (t_2 - t_1)/2$, $h'_N = (t_N - t_{N-1})/2$, $h'_l = (t_{l+1} - t_{l-1})/2$ for $l = 2,\ldots,N-1$, $\vec{\gamma} = (\gamma_l)_{l=1}^{N}$, $\gamma_1 = 0, \gamma_N = 1$, $\gamma_l = \gamma(t_l)$, $\dot{\gamma}_l = (\gamma_{l+1} - \gamma_l)/h_l$ for $l = 1,\ldots,N-1$, $\dot{\gamma}_N = \dot{\gamma}_1$, $h_l = t_{l+1} - t_l$ for $l = 1,\ldots,N-1$, and $q_2(\gamma_l)$, $l = 1,\ldots,N$, are approximations of q_2 at each γ_l obtained from the interpolation of $q_2(t_l)$, $l = 1,\ldots,N$, by a cubic spline. Accordingly, for the purpose of minimizing (11), we use a procedure that alternates computations, as described in Section 2 and Section 3, of approximately optimal diffeomorphisms (a constant number of them per iteration for reparametrizing the second curve) and successive computations of approximately optimal rotation matrices (for rotating the first curve) for all starting points of the first curve. As noted in [4], carrying out computations this way is not only more efficient all by itself, but, if both curves are closed, allows applications of the Fast Fourier Transform (FFT) as demonstrated in [5] for $d = 2$, for computing successively in an even more efficient manner, as described in the next section, optimal rotation matrices for all starting points of the first curve. The procedure, with K, $q_1(t_l)$, $q_2(t_l)$, $l = 1,\ldots,N$, as input, is summarized in Procedure 2 below. Note, $itop$ in the procedure is an input variable that must be set equal to a positive integer that is constant relative to N, and not larger than the cardinality of K. It is the number of times the second **for** loop in the **repeat** loop of the procedure is executed during each iteration of the **repeat** loop. It is in the second **for** loop that the DP algorithm is executed, thus the execution time of the procedure can be large if $itop$ is greater than 1. Actually the first **for** loop in the **repeat** loop of the procedure takes a lot less time than the second **for** loop even if $itop$ equals 1. It is in the first **for** loop that the KU algorithm is executed. We note, in our experiments, $itop$ equal to 1 has usually sufficed for curves of relatively simple curvature, e.g., spherical ellipsoids in $3-$dimensional space (see Section 6). For curves of more complex curvatures, higher values have usually been required for the successful execution of the procedure.

Procedure 2

Set $h_l = t_{l+1} - t_l$ for $l = 1, \ldots, N-1$.
Set $h'_1 = (t_2 - t_1)/2$, $h'_N = (t_N - t_{N-1})/2$, $h'_l = (t_{l+1} - t_{l-1})/2$ for
$l = 2, \ldots, N-1$.
Set $\hat{q}_2(t_l) = q_2(t_l)$ for $l = 1, \ldots, N$.
Set $iter = 0$, $E^{curr} = 10^6$, $iten = 10$, $tol = 10^{-6}$.
repeat
  Set $iter = iter + 1$, $E^{prev} = E^{curr}$.
  for each $t_0 \in K$ **do**
    Set $\hat{q}_1(t_l) = q_1(t_0 + t_l)$ for $l = 1, \ldots, N$.
    Execute KU algorithm for $\hat{q}_2(t_l)$, $\hat{q}_1(t_l)$, $l = 1, \ldots, N$, to get rotation matrix R and $maxtrace$.
    Identify (t_0, R) as a couple of interest and associate with it the value of $maxtrace$.
    Keep track of identified couples of interest (t_{0i}, R_i), $i = 1, \ldots, itop$, satisfying that for each i, $i = 1, \ldots, itop$, the value of $maxtrace$ associated with (t_{0i}, R_i) is one of the $itop$ largest values among the values of $maxtrace$ associated with all couples of interest identified so far.
  end for
  for $i = 1, \ldots, itop$ **do**
    Set $t_0 = t_{0i}$, $R = R_i$.
    Set $\hat{q}_1(t_l) = Rq_1(t_0 + t_l)$ for $l = 1, \ldots, N$.
    Execute DP algorithm for $\hat{q}_1(t_l)$, $q_2(t_l)$, $l = 1, \ldots, N$, to get
    discretized diffeomorphism $\vec{\gamma} = (\gamma_l)_{l=1}^N$.
    Set $\dot{\gamma}_l = (\gamma_{l+1} - \gamma_l)/h_l$ for $l = 1, \ldots, N-1$, $\dot{\gamma}_N = \dot{\gamma}_1$.
    From interpolation of $q_2(t_l)$, $l = 1, \ldots, N$, with a cubic spline
    set $\hat{q}_2(t_l) = \sqrt{\dot{\gamma}_l} q_2(\gamma_l)$ for $l = 1, \ldots, N$.
    Compute $E^{curr} = E^{discr}(t_0, R, \vec{\gamma}) = \sum_{l=1}^N h'_l \, \| Rq_1(t_0 + t_l) - \sqrt{\dot{\gamma}_l} q_2(\gamma_l) \|^2$
    $= \sum_{l=1}^N h'_l \, \| \hat{q}_1(t_l) - \hat{q}_2(t_l) \|^2$.
    Keep track of triple $(t_0, R, \vec{\gamma})$, E^{curr}, $\hat{q}_1(t_l)$, $\hat{q}_2(t_l)$, $l = 1, \ldots, N$, with smallest value for E^{curr}.
  end for
until $|E^{curr} - E^{prev}| < tol$ or $iter > iten$.
From interpolation of $\beta_2(t_l)$, $l = 1, \ldots, N$, with a cubic spline
set $\hat{\beta}_2(t_l) = \beta_2(\gamma_l)$ for $l = 1, \ldots, N$.
Set $\hat{\beta}_1(t_l) = R\beta_1(t_0 + t_l)$ for $l = 1, \ldots, N$.
Return $(t_0, R, \vec{\gamma})$, $E^{discr}(t_0, R, \vec{\gamma})$ $(= E^{curr})$, $\hat{\beta}_1(t_l)$, $\hat{q}_1(t_l)$, $\hat{\beta}_2(t_l)$, $\hat{q}_2(t_l)$,
$l = 1, \ldots, N$.

On output, $E^{discr}(t_0, R, \vec{\gamma})$ is interpreted to be the elastic shape distance between β_1 and β_2. On the other hand, $\hat{\beta}_1(t_l)$ and $\hat{\beta}_2(t_l)$, $l = 1, \ldots, N$, are interpreted to achieve the elastic registration of β_1 and β_2.

Similar to Procedure 1, Procedure 2 above is also set up to handle the particular case in which d equals 1 and neither curve is closed. But since similar to Procedure 1 the requirement that the two curves β_1 and β_2 must have the same number of points with the same partition, is not necessary for this case, Procedure 1' can be used instead of Procedure 2 in the absence of the requirement.

5. Successive Computations of Rotations with FFT for Rigid Alignment of Curves

Again, let β_1, β_2, q_1, q_2 be as above, i.e., $\beta_n : [0,1] \to \mathbb{R}^d$, $n = 1,2$, are absolutely continuous functions representing simple curves in $\mathbb{R}^d$ of unit length, and $q_n : [0,1] \to \mathbb{R}^d$, $n = 1,2$, are square-integrable functions that are the shape functions or SRVF's of β_n, $n = 1,2$, respectively. In this section, using arguments similar to those used in [5] for $d = 2$, we first present an alternative version of the KU algorithm that uses the FFT for the purpose of speeding up the successive computations in Procedure 2 in the previous section, of approximately optimal rotation matrices for all starting points of one of the curves. These computations actually take place in the first **for** loop of that procedure. Taking into account the nature of the FFT, we assume both curves are closed (this will become evident below), and without any loss of generality, for the purpose of developing the alternative version of the KU algorithm in a manner similar to the way in which the KU algorithm was developed in Section 3, assume that any point in β_2 can be treated as a starting point of β_2, and that it is β_1 that has a fixed starting point. In particular, it follows then that q_2 can be interpreted to be a periodic function from $\mathbb{R}$ into $\mathbb{R}^d$, $q_2(t+1) = q_2(t)$ for all vaues of t. Taking into account as well the part of Procedure 2 in the previous section that we are trying to improve (the first **for** loop), and following the reasoning in Section 3 to obtain (4), ideally, we would like to solve a problem of the following type: Find $t_0 \in [0,1]$, and a $d \times d$ rotation matrix R that minimize

$$E(t_0, R) = \int_0^1 \|q_1(t) - Rq_2(t_0 + t)\|^2 dt. \tag{12}$$

As noted above, in practice, we work with curves β_1, β_2, given as discrete sets of points. Accordingly, for some integer $N > 0$, and a partition of $[0,1]$, $\{t_l\}_{l=1}^N$, $t_1 = 0 < t_2 < \ldots < t_N = 1$, for $n = 1,2$, the curve β_n is given as a list of N points in the curve, where for $l = 1, \ldots, N$, $\beta_n(t_l)$ is the l^{th} point in the list for β_n. Similarly for q_1, q_2, except that for $l = 1, \ldots, N$, $q_1(t_l)$ and $q_2(t_l)$ are computed as described in Section 2. Again, we assume K as defined in the previous section equals $\{t_1, \ldots, t_{N-1}\}$ or a subset of it, a subset essentially of size $O(N)$, so that $\{\beta_2(t), t \in K\}$ is then interpreted to be the set of starting points of β_2. Also, as justified in the previous section, the partition $\{t_l\}_{l=1}^N$ must then be uniform. In what follows, for $l = 1, \ldots, N$, $k = 1, \ldots, d$, $j = 1, \ldots, d$, q_1^l is $q_1(t_l)$, q_2^l is $q_2(t_l)$, q_{1k}^l is the k^{th} coordinate of q_1^l, q_{2j}^l is the j^{th} coordinate of q_2^l, and $\hat{q}^l_{1k}$ is q_{1k}^{N-l+1}.

In order to discretize integral (12), we define for each $m = 1,\ldots,N-1$, points $q_2^{m\oplus l}, l = 1,\ldots,N$, by

$$q_2^{m\oplus l} = q_2(t_m + t_l),$$

and let $q_{21}^{m\oplus l},\ldots,q_{2d}^{m\oplus l}$ be the d coordinates of $q_2^{m\oplus l}$ so that

$$(q_{21}^{m\oplus l},\ldots,q_{2d}^{m\oplus l})^T = q_2^{m\oplus l}.$$

We note as well that for $m = 1,\ldots,N-1$, we may then assume the existence of additional functions $q_2^m : [0,1] \to \mathbb{R}^d$, given in their discretized form as

$$q_2^m(t_l) = q_2(t_m + t_l) = q_2^{m\oplus l}, l = 1,\ldots,N.$$

With $1 \le m \le N-1$, letting $h = 1/(N-1)$, we then discretize integral (12) using the uniform trapezoidal rule for when both curves are closed:

$$E^{discr}(m,R) = h\sum_{l=1}^{N-1} \|q_1(t_l) - Rq_2^m(t_l)\|^2 = h\sum_{l=1}^{N-1} \|q_1^l - Rq_2^{m\oplus l}\|^2. \tag{13}$$

Thus, the problem of finding $t_0 \in [0,1]$ and a $d\times d$ rotation matrix R that minimize (12) becomes the problem of finding m, $1 \le m \le N-1$, with t_m in K, and a $d\times d$ rotation matrix R that minimize (13).
For this purpose, for each m, $m = 1,\ldots,N-1$, we define a $d\times d$ matrix $A(m)$ by defining its entries $A_{kj}(m)$ for each pair $k, j = 1,\ldots,d$, by

$$A_{kj}(m) = \sum_{l=1}^{N-1} q_{1k}^l q_{2j}^{m\oplus l}, \tag{14}$$

so that for fixed m, minimizing (13) over all $d\times d$ rotation matrices R is equivalent to maximizing

$$\sum_{l=1}^{N-1} (q_1^l)^T R q_2^{m\oplus l} = \mathrm{tr}(RA(m)^T). \tag{15}$$

We note that for fixed m, we can execute the KU algorithm for $q_1(t_l)$, $q_2^m(t_l)$, $l = 1,\ldots,N$, to compute R that maximizes (15). Doing this for each m, $1 \le m \le N-1$, with t_m in K, we identify among them an m for which the maximization of (15) is the largest. The solution is then that m together with the rotation matrix R at which the maximization is achieved. We also note that computing $A(m)$ for each m is $O(N)$ so that computing $O(N)$ of them is then $O(N^2)$ if each $A(m)$ is computed separately. This is exactly how it is done in Procedure 2 in the previous section.

For each pair $k, j = 1,\ldots,d$, with $A_{kj}(m)$ as in (14), we propose to compute all of $A_{kj}(1),\ldots,A_{kj}(N-1)$ in $O(N\log N)$ time using the FFT to accomplish the Discrete Fourier Transform (DFT). For this purpose, for $k = 1,\ldots,d$, let $\hat{q}_{1k} = (\hat{q}_{1k}^1,\ldots,\hat{q}_{1k}^{N-1})$, and for $j = 1,\ldots,d$, let $q_{2j} = (q_{2j}^1,\ldots,q_{2j}^{N-1})$. Given arbitrary vectors x, y of length $N-1$, we let

$\mathbf{DFT}(x)$ and $\mathbf{DFT}^{-1}(y)$ denote the DFT of x and the inverse DFT of y, respectively. With the symbol $\cdot$ indicating component by component multiplication of two vectors, then by the convolution theorem for the DFT we have for each pair $k, j = 1, \ldots, d$,

$$\begin{aligned}(A_{kj}(1), \ldots, A_{kj}(N-1)) &= (\sum_{l=1}^{N-1} q_{1k}^{l} q_{2j}^{1 \oplus l}, \ldots, \sum_{l=1}^{N-1} q_{1k}^{l} q_{2j}^{(N-1) \oplus l}) \\ &= \mathbf{DFT}^{-1}[\mathbf{DFT}(\hat{q}_{1k}) \cdot \mathbf{DFT}(q_{2j})]\end{aligned}$$

which for each pair $k, j = 1, \ldots, d$, enables us to reduce the computation of all of $A_{kj}(1), \ldots, A_{kj}(N-1)$ to three $O(N \log N)$ FFT operations. Thus, we can compute all of $A(1), \ldots, A(N-1)$ in $O(N \log N)$ time with the FFT.

An outline of the alternative version of the KU algorithm, the KU2 algorithm, that uses the FFT, follows. Here for arbitrary vectors x, y of length $N-1$, $\mathbf{FFT}(x)$, $\mathbf{IFFT}(y)$ denote $\mathbf{DFT}(x)$, $\mathbf{DFT}^{-1}(y)$, respectively, computed with the FFT. Note, *itop* in the algorithm is an input variable as described in the previous section before the outline of Procedure 2, an input variable used there exclusively in that procedure, its purpose to control the number of times the DP algorithm is executed in the procedure. Here, with the same purpose, before it is an input variable of the KU2 algorithm, it is first an input variable of a procedure in which the KU2 and DP algorithms are executed, Procedure 3, the outline of which appears later in this section, its purpose discussed as well.

Algorithm Kabsch-Umeyama with FFT (KU2 algorithm)

Set q_{1k}^{l} equal to k^{th} coordinate of $q_1(t_l)$ for $l = 1, \ldots, N$, $k = 1, \ldots, d$.
Set q_{2j}^{l} equal to j^{th} coordinate of $q_2(t_l)$ for $l = 1, \ldots, N$, $j = 1, \ldots, d$.
Set $\hat{q}_{1k}^{l} = q_{1k}^{N-l+1}$ for $l = 1, \ldots, N$, $k = 1, \ldots, d$.
Set $\hat{q}_{1k} = (\hat{q}_{1k}^{1}, \ldots, \hat{q}_{1k}^{N-1})$ for $k = 1, \ldots, d$.
Set $q_{2j} = (q_{2j}^{1}, \ldots, q_{2j}^{N-1})$, for $j = 1, \ldots, d$.
for each pair $k, j = 1, \ldots, d$ **do**
  Compute $(A_{kj}(1), \ldots, A_{kj}(N-1)) = \mathbf{IFFT}[\mathbf{FFT}\ (\hat{q}_{1k}) \cdot\ \mathbf{FFT}(q_{2j})]$.
end for
for each m, $1 \leq m \leq N-1$, with $t_m \in K$ **do**
  Identify $d \times d$ matrix $A(m)$ with entries $A_{kj}(m)$ for
  each pair $k, j = 1, \ldots, d$.
  Compute SVD of $A(m)$, i.e., identify $d \times d$ matrices U, S, V,
  so that $A(m) = USV^T$ in the SVD sense.
  Set $s_1 = \ldots = s_{d-1} = 1$.
  if $\det(UV) > 0$ **then** set $s_d = 1$.
  else set $s_d = -1$. **end if**
  Set $\tilde{S} = \mathrm{diag}\{s_1, \ldots, s_d\}$.
  Compute $R = U\tilde{S}V^T$ and $maxtrace = \mathrm{tr}(RA(m)^T)$.
  Identify (m, R) as a couple of interest and associate with it the value of *maxtrace*.

Keep track of identified couples of interest (m_i, R_i), $i = 1, \ldots, itop$, satisfying that for each i, $i = 1, \ldots, itop$, the value of *maxtrace* associated with (m_i, R_i) is one of the $itop$ largest values among the values of *maxtrace* associated with all couples of interest identified so far.

end for

Return couples (m_i, R_i), $i = 1, \ldots, itop$.

We note that if $d = 1$, the KU2 algorithm still computes couples (m_i, R_i), $i = 1, \ldots, itop$, with the resulting R_i's always equal to 1.

Finally, a modified version of Procedure 2 in the previous section, Procedure 3, follows. Here β_1, β_2, q_1, q_2, N, $\{t_l\}_{l=1}^N$, $\beta_1(t_l)$, $\beta_2(t_l)$, $q_1(t_l)$, $q_2(t_l)$, $l = 1, \ldots, N$, are as above. Thus, β_1, β_2 are closed curves and $\{t_l\}_{l=1}^N$ is uniform. The procedure with K, $itop$, $q_1(t_l)$, $q_2(t_l)$, $l = 1, \ldots, N$, as input, is essentially the same as Procedure 2, except that the **for** loop in Procedure 2 that executes the KU algorithm of Section 3 as many times as there are starting points of the first curve (β_1), is replaced by the execution of the KU2 algorithm outlined above. This has the effect of speeding up the successive computations appearing in Procedure 2 of approximately optimal rotation matrices for all starting points of the first curve due to the fact that the KU2 algorithm uses the FFT which takes $O(N \log N)$ time, while the **for** loop in Procedure 2 computes each approximately optimal rotation matrix separately thus taking $O(N^2)$ time.

Procedure 3

Set $h = 1/(N-1)$.
Set $\hat{q}_2(t_l) = q_2(t_l)$ for $l = 1, \ldots, N$.
Set $iter = 0$, $E^{curr} = 10^6$, $iten = 10$, $tol = 10^{-6}$.
repeat
  Set $iter = iter + 1$, $E^{prev} = E^{curr}$.
  Execute KU2 algorithm for $\hat{q}_2(t_l)$, $q_1(t_l)$, $l = 1, \ldots, N$, to get
  couples (m_i, R_i), m_i an integer, $1 \leq m_i \leq N-1$, R_i a rotation matrix, $i = 1, \ldots, itop$.
  for $i = 1, \ldots, itop$ **do**
    Set $m = m_i$, $R = R_i$.
    Set $\hat{q}_1(t_l) = R q_1(t_m + t_l)$ for $l = 1, \ldots, N$.
    Execute DP algorithm for $\hat{q}_1(t_l)$, $q_2(t_l)$, $l = 1, \ldots, N$, to get
    discretized diffeomorphism $\vec{\gamma} = (\gamma_l)_{l=1}^N$.
    Set $\dot{\gamma}_l = (\gamma_{l+1} - \gamma_l)/h$ for $l = 1, \ldots, N-1$, $\dot{\gamma}_N = \dot{\gamma}_1$.
    From interpolation of $q_2(t_l)$, $l = 1, \ldots, N$, with a cubic spline
    set $\hat{q}_2(t_l) = \sqrt{\dot{\gamma}_l} q_2(\gamma_l)$ for $l = 1, \ldots, N$.
    Compute $E^{curr} = E^{discr}(t_m, R, \vec{\gamma})$
    $= \sum_{l=1}^{N-1} h \, \| R q_1(t_m + t_l) - \sqrt{\dot{\gamma}_l} q_2(\gamma_l) \|^2 = \sum_{l=1}^{N-1} h \, \| \hat{q}_1(t_l) - \hat{q}_2(t_l) \|^2$.

Keep track of triple $(t_m, R, \vec{\gamma})$, E^{curr}, $\hat{q}_1(t_l)$, $\hat{q}_2(t_l)$, $l = 1, \ldots, N$, with smallest value for E^{curr}.

end for

until $|E^{curr} - E^{prev}| < tol$ or $iter > iten$.

From interpolation of $\beta_2(t_l)$, $l = 1, \ldots, N$, with a cubic spline

set $\hat{\beta}_2(t_l) = \beta_2(\gamma_l)$ for $l = 1, \ldots, N$.

Set $\hat{\beta}_1(t_l) = R\beta_1(t_m + t_l)$ for $l = 1, \ldots, N$.

Return $(t_m, R, \vec{\gamma})$, $E^{discr}(t_m, R, \vec{\gamma})$ $(= E^{curr})$, $\hat{\beta}_1(t_l)$, $\hat{q}_1(t_l)$, $\hat{\beta}_2(t_l)$, $\hat{q}_2(t_l)$, $l = 1, \ldots, N$.

On output, $E^{discr}(t_m, R, \vec{\gamma})$ is interpreted to be the elastic shape distance between β_1 and β_2. On the other hand, $\hat{\beta}_1(t_l)$ and $\hat{\beta}_2(t_l)$, $l = 1, \ldots, N$, are interpreted to achieve the elastic registration of β_1 and β_2.

6. Results from Computations with Implementation of Methods

A software package that incorporates the methods presented in this paper for computing the elastic registration of two simple curves in $d-$dimensional space, d a positive integer, and therefore the elastic shape distance between them, has been implemented. The implementation is in Matlab[1] with the exception of the Dynamic Programming routine which is written in Fortran but is executed as a Matlab mex file. In this section, we present results obtained from executions of the software package with $d = 3$. We note, the sofware package as well as input data files, a README file, etc. can be obtained at the following link

`https://doi.org/10.18434/mds2-2329`

With the exception of the Matlab driver routine, ESD_driv_3 dim.m, which is designed for the case $d = 3$, all Matlab routines in the package can be executed for any d if the current driver routine is adjusted or replaced to handle the value of d. However, parameter dimx in the Fortran routine DP_MEX_WNDSTRP_ALLDIM.F may have to be modified so that instead of having a value of 3, it has the value of d. The Fortran routine must then be processed to obtain a new mex file for the routine by typing in the Matlab window: mex -compatibleArrayDims DP_MEX_WNDSTRP_ALLDIM.F

Given discretizations of two simple curves β_1, β_2, $\beta_1 : [0, T_1] \to \mathbb{R}^d$, $\beta_2 : [0, T_2] \to \mathbb{R}^d$, $T_1,\ T_2 > 0$, the elastic registration of β_1 and β_2 to be computed together with the elastic shape distance between them, irrespective of the value of d, the program always proceeds first to compute an approximation of the length of each curve by computing the length of each line segment joining consecutive points on the curve in the discretization of the curve and adding these lengths, and then proceeds to scale the two curves so that each curve has approximate length 1 (each point in the discretization of each curve is divided by the approximate length of the curve). The program then proceeds to scale, if any, the two

[1]The identification of any commercial product or trade name does not imply endorsement or recommendation by the National Institute of Standards and Technology.

partitions that discretize the curves so that they become partitions of $[0,1]$, or if no partitions are given, to create two partitions of $[0,1]$, one for each curve, according to the number of points in the discretization of each curve, the discretization of each curve then assumed to be the result of discretizing the curve by the corresponding partition, each partition uniform if at least one curve is closed, each partition parametrizing the corresponding curve by arc length otherwise. Utilizing the given or created partitions and the discretizations of the curves, with the exception of the case in which d equals 1 and both curves are open, the program then proceeds to create a common partition of $[0,1]$ for the two curves and to discretize each curve by this common partition using cubic splines. If at least one curve is closed, and the numbers of points in the first curve and second curve are N and M, respectively, letting L equal the larger of N and M, the common partition is then taken to be the uniform partition of $[0,1]$ of size equal to L. This is in accordance with the requirement established in Section 4 that if at least one curve is closed (the set of starting points of one of the curves will have more than one point), in order to compute the elastic shape distance and registration in the appropriate manner, the curves should be discretized by the same uniform partition. Note that a set of starting points of one of the curves having more than one point is then identified satisfying that it is the discretization by the uniform partition of one of the curves (a closed curve), or a subset of it. On the other hand, if both curves are open, d not equal to 1, the common partition is then taken to be the union of the two partitions discretizing the curves minus certain points in this union that are eliminated systematically so that the distance between any two consecutive points in the common partition does not exceed some tolerance. This is in accordance with the requirement established in Section 4 that if both curves are open (each curve has exactly one starting point), d not equal to 1, in order to compute the elastic shape distance and registration in the appropriate manner, the curves should be discretized by the same partition, a partition not necessarily uniform. Finally, if both curves are open and $d = 1$, no common partition is created and the curves continue to be discretized by the same given or created partitions. All of the above is accomplished by Matlab routine ESD_comp_alldim.m during the execution of the software package. Once this routine is done, the actual computations of the elastic shape distance and registration are carried out by Matlab routine ESD_core_alldim.m in which all of the procedures presented in Section 4 and Section 5 have been implemented.

The results that follow were obtained from applications of our software package on discretizations of curves in $3-$dimensional space of the helix and spherical ellipsoid kind. With an observer at the origin of the $3-$dimensional Euclidean space whose line of sight is the positive $z-$axis, given $T > 0$, a circular helix of radius 1 with axis of rotation the positive $z-$axis and that moves away from the observer in a clockwise screwing motion, is defined by

$$x(t) = \cos t, \quad y(t) = \sin t, \quad z(t) = t, \quad t \in [0,T].$$

On the other hand, with an observer at the origin of the $3-$dimensional Euclidean space whose line of sight is the positive $z-$axis, given r,a,b, with $r > a > 0$, $r > b > 0$, a spherical ellipsoid with axis of rotation the positive $z-$axis and that as viewed by the observer moves

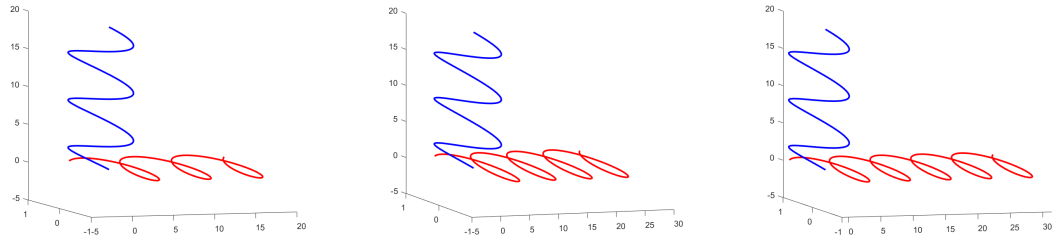

Fig. 3. Three plots of helices. The elastic registration of the two helices in each plot and the elastic shape distance between them were computed.

around its axis of rotation in a clockwise direction, is defined by

$$x(t) = a\cos t, \quad y(t) = b\sin t, \quad z(t) = (r^2 - x(t)^2 - y(t)^2)^{1/2}, \quad t \in [0, 2\pi].$$

We note that in the obvious similar manner, helices and spherical ellipsoids with axis of rotation the positve/negative $x-, y-, z-$axis can be defined as well. We also note that as defined above helices are open curves, and spherical ellipsoids are closed curves.

Three plots depicting helices are shown in Figure 3. (Note that in the plots there, the $x-$axis, the $y-$axis and the $z-$axis are not always to scale relative to one another). In each plot two helices appear. The helix in each plot with the positive $z-$axis as its axis of rotation was considered to be the first curve or helix in the plot. In each plot this helix was obtained by setting T to 6π in the definiton of a helix above so that it has three loops in each plot and thus is the same helix in all three plots. The other helix in each plot has the positive $x-$axis as its axis of rotation and was considered to be the second curve or helix in each plot. From left to right in the three plots, the second helix was obtained by setting T to 6π, 8π, 10π, respectively, in the definition of a helix above, the definition modified in the obvious manner so that the helix has the positive $x-$axis as its axis of rotation. Thus the second helix has three, four, five loops, from left to right in the three plots. All helices in the plots were then discretized as described below and the elastic registration of the two helices in each plot and the elastic shape distance between them were then computed through executions of our software package (mostly executions of Procedure 2 in Section 4). Accordingly, one would expect the elastic shape distances, if given in the order of the plots from left to right, to have been in strictly increasing order with the first distance essentially equal to zero. That is exactly what we obtained: 0.00000 0.48221 0.60352.

We note that on input the first helix in each plot was given as the same discretization of a $3-$loop helix, a helix discretized by a uniform partition of $[0, 6\pi]$, the discretization consisting of 451 points. On the other hand, the second helix in each plot was given as well as the discretization of a helix, from left to right in the three plots a helix having 3, 4, 5 loops, respectively, a helix discretized by a uniform partition of $[0, 6\pi]$, $[0, 8\pi]$, $[0, 10\pi]$, respectively, the discretizations consisting of 451, 601, 751 points, respectively. Given a pair of helices in one of the three plots, as described above for the case in which neither curve is closed, $d \neq 1$, the program then, after scaling each helix in the pair to have approximate

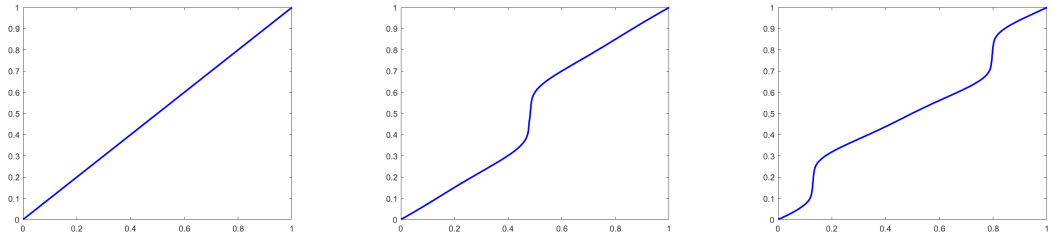

Fig. 4. Optimal diffeomorphisms for pairs of helices.

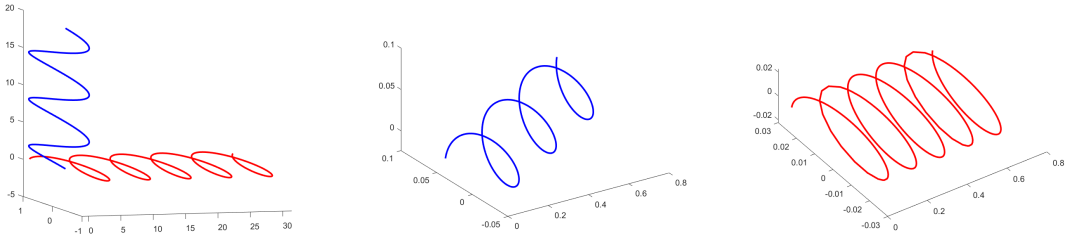

Fig. 5. Views of first helix of 3 loops and second helix of 5 loops before computation of elastic shape distance and registration (left), of rotated first helix (middle) and reparametrized second helix (right) after computations.

length 1 and scaling the partition discretizing each helix to be a partition of $[0,1]$, created a common partition of $[0,1]$ for the two helices, a nonuniform partition, and discretized each helix by the common partition using cubic splines. From left to right in the three plots, the common partitions were of size 451, 901, 1051, respectively. We note as well that in each case we assumed (correctly) the helices to be defined in the proper directions (see second paragraph of the Introduction section), thus cutting the times of execution for each case by about half. For each case from left to right in the three plots, the times of execution were 5.4, 19.4, 39.2 seconds, respectively, with the **repeat** loop in Procedure 2 in Section 4 executed 2, 3, 5 times, respectively. Plots of the computed optimal diffeomorphisms for each pair of helices from left to right in the three plots in Figure 3, are shown in Figure 4. The computed optimal rotation matrix for the pair in the leftmost plot in Figure 3, was $\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$. For each of the other two pairs it was almost the same matrix, the entries slightly different. Finally, Figure 5 shows results of the elastic registration of the helix of 3 loops and the helix of 5 loops. The two helices are shown in the leftmost plot of the figure before any computations took place. In the middle plot we see the first helix (of 3 loops) after it was rotated with the computed optimal rotation matrix, its axis of rotation becoming a ray of direction not far from that of the positive $x-$axis. In the rightmost plot we see the second helix (of 5 loops) after it was reparametrized with the computed optimal diffeomorphism, some of the consecutive points in its discretization becoming slightly separated, in particular near the end of the first loop and the beginning of the fifth loop, so that the plot of the

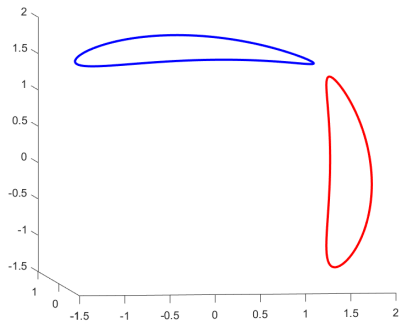

Fig. 6. Two spherical ellipsoids, curves in 3−d space. The positive $z-$axis is the axis of rotation of one spherical ellipsoid, while the positive $x-$axis is the axis of rotation of the other one. Their shapes are essentially identical thus the elastic shape distance between them should be essentially zero.

helix, which is drawn by joining with line segments consecutive points in the discretization of the helix, has a slighly flat appearance in these areas.

Finally, we note that we could generate results using spherical ellipsoids similar to the results just presented for helices. Since such an exercise is tantamount to repeating what has already been done, in its place, we have opted to use spherical ellipsoids for the purpose of illustrating, if the curves under consideration are closed, the improvement in execution time that is achieved through the execution of our software package when it involves the FFT (mostly the execution of Procedure 3 in Section 5) as this has the effect of speeding up the successive computations appearing in Procedure 2 in Section 4 of optimal rotation matrices for all starting points of one of the curves. We also use spherical ellipsoids to illustrate what occurs if the curves under consideration are not defined in the proper directions.

A plot depicting two spherical ellipsoids of essentially the same shape is shown in Figure 6. The ellipsoid with the positive $z-$axis as its axis of rotation was considered to be the first curve or ellipsoid in the plot. It was obtained by setting $r = 2.0$, $a = 1.3$, $b = 1.0$ in the definition of a spherical ellipsoid above. The other ellipsoid in the plot has the positive $x-$axis as its axis of rotation and was considered to be the second curve or ellipsoid in the plot. It was obtained by setting $r = 2.0$, $a = 1.0$, $b = 1.3$ in the definition of a spherical ellipsoid above, the definition modified in the obvious manner so the the ellipsoid has the positive $x-$axis as its axis of rotation. The two ellipsoids in the plot were then discretized as described below, and taking into account that both are closed, the elastic registration of the two ellipsoids and the elastic shape distance (essentially zero) between them were then successfully computed through the execution of our software package, first without involving the FFT thus executing mostly Procedure 2 in Section 4, and then involving the FFT thus executing mostly Procedure 3 in Section 5. Note that for both procedures the input variable $itop$ was set to 1 as suggested in Section 4 for spherical ellipsoids.

First we discretized the first ellipsoid by a nonuniform partition of $[0,2\pi]$ of size 1001 and the second ellipsoid by a uniform partition of $[0,2\pi]$ of size 901. As described above

for the case in which at least one curve is closed, the program then, after scaling each ellipsoid to have approximate length 1 and scaling the partition discretizing each curve to be a partition of $[0, 1]$, created a common partition of $[0, 1]$ for the two ellipsoids, a uniform partition of size 1001, and discretized each curve by this common partition using cubic splines. The program then selected the discretization of the first ellipsoid by the uniform partition as the set of starting points of this ellipsoid. Without involving the FFT, the **repeat** loop in Procedure 2 was executed two times, i.e., there was a total of two iterations for this loop. The same for the **repeat** loop in Procedure 3 when involving the FFT. Without the FFT, the executions of the KU algorithm for computing successively optimal rotation matrices for all starting points of the first ellipsoid, took about 0.12 seconds per iteration of the **repeat** loop, while the execution of the DP algorithm took about 6.5 seconds. With the FFT, the execution of the KU2 algorithm, again for computing successively optimal rotation matrices for all starting points of the first ellipsoid, took about 0.06 seconds per iteration, while the DP algorithm took about 6.5 seconds.

Replacing above 1001 by 46001 and 901 by 45001, and repeating exactly what was done as described above, we then obtained that without the FFT, the executions of the KU algorithm took about 87 seconds per iteration of the **repeat** loop in Procedure 2 (two iterations), while the execution of the DP algorithm took about 291 seconds, and with the FFT, the execution of the KU2 algorithm took about 2 seconds per iteration of the **repeat** loop in Procedure 3 (two iterations), while the DP algorithm took about 291 seconds.

From the two examples above it is clear that computing successively optimal rotation matrices for all starting points of the first ellipsoid with the FFT is a lot faster than without it. The two examples illustrate as well the linearity of the DP algorithm and that its execution time appears to be significantly larger than the time required to compute successively in either Procedure 2 or Procedure 3, optimal rotation matrices for all starting points of the first ellipsoid. Although the latter may be true when the FFT is used, it is not exactly true otherwise. Actually as the size of the discretizations of the curves increases, if the FFT is used, this time becomes insignificant relative to the execution time of the DP algorithm, but the opposite occurs if it is not.

Finally we reversed the direction of the first ellipsoid in the last example above and as expected obtained an elastic shape distance between the two ellipsoids different from zero, a distance of 0.195. Using the option in the program to do the computations in both directions of one of the curves, we then obtained the correct distance (essentially zero). Of course the execution time of the program doubled.

7. Summary

Inspired by Srivastava et al.'s work for computing the elastic registration of two simple curves in $d-$dimensional Euclidean space, d a positive integer, and thus the associated elastic shape distance between them, in this paper we have enhanced Srivastava et al.'s work in various ways. First we have presented a Dynamic Programming algorithm that is linear for computing an optimal diffeomorphism for the elastic registration of two simple curves in $d-$dimensional space, the computation of the registration based only on reparametrizations (with diffeomorphisms of the unit interval) of one of the curves (no rotations), the curves given on input as discrete sets of nodes in the curves, the numbers of nodes in the curves not necessarily equal, the partitions of the unit interval discretizing the curves not necessarily uniform. Next we have presented a purely algebraic justification of the usual algorithm, the Kabsch-Umeyama algorithm, for computing an optimal rotation matrix for the rigid alignment of two simple curves in $d-$dimensional space, the curves given on input as discrete sets of nodes in the curves, the same number of nodes in each curve, the two curves discretized by the same partition of the unit interval, the partition discretizing the curves not necessarily uniform. Lastly, with the convention that if one of the curves is closed, the first curve is closed, we have redefined the L^2 type distance that is minimized in Srivastava et al.'s work to allow for the second curve to be reparametrized while the first one is rotated, the curves again given on input as discrete sets of nodes in the curves, the same number of nodes in each curve, both curves now discretized by the same partition of the unit interval (a uniform partition if the first curve is closed). A finite subset of the nodes in the first curve (possibly all of them, possibly one if neither curve is closed) is then selected which we interpret to be the set of so-called starting points of the curve, and the redefined L^2 type distance is then minimized with an iterative procedure that alternates computations of optimal diffeomorphisms (a constant number of them per iteration for reparametrizing the second curve) with successive computations of optimal rotation matrices (for rotating the first curve) for all starting points of the first curve. Carrying out computations this way is not only more efficient all by itself, but, if both curves are closed, allows applications of the Fast Fourier Transform (FFT) for computing successively in an even more efficient manner, optimal rotation matrices for all starting points of the first curve. We note, results from computations with the implementation of our methods applied on $3-$dimensional curves of the helix and spherical ellipsoid kind, have been presented in this paper as well.

References

[1] Bernal, J., Dogan, G., Hagwood, C. R.: Fast Dynamic Programming for Elastic Registration of Curves. Proceedings of DIFF-CVML workshop, CVPR 2016, Las Vegas, Nevada. (2016)

[2] Bernal, J.: Shape Analysis, Lebesgue Integration and Absolute Continuity Connections. NISTIR 8217 (2018).

[3] Bernal, J., Lawrence, J.: Characterization and Computation of Matrices of Maximal Trace Over Rotations. Journal of Geometry and Symmetry in Physics. 53 (2019).

[4] Dogan, G., Bernal, J., Hagwood, C. R.: A Fast Algorithm for Elastic Shape Distances between Closed Planar Curves. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA. June 2015.

[5] Dogan, G., Bernal, J., Hagwood, C. R.: FFT-based alignment of 2d closed curves with application to elastic shape analysis. Proceedings of the 1st DIFF-CV Workshop, British Machine Vision Conference, Swansea, Wales, UK. September 2015.

[6] Kabsch, W.: A solution for the best rotation to relate two sets of vectors. Acta Crystallographica Section A: Crystal Physics. 32(5): 922-923 (1976).

[7] Kabsch, W.: A discussion of the solution for the best rotation to relate two sets of vectors. Acta Crystallographica Section A: Crystal Physics. 34(5): 827-828 (1978).

[8] Karypis, G., Aggarwal, R., Kumar, V., Shekhar, S.: Multilevel hypergraph partitioning: Applications in VLSI domain. Proc. of the Design and Automation Conference. (1997)

[9] Lawrence, J., Bernal, J., Witzgall, C.: A Purely Algebraic Justification of the Kabsch-Umeyama Algorithm. Journal of Research of the National Institute of Standards and Technology. 124 (2019).

[10] Lay D., Lay S. and McDonald J., *Linear Algebra and its Applications,* 5th edition, Pearson Education, Boston 2016.

[11] Markley, F. L.: Equivalence of Two Solutions of Wahba's Problem. Journal of the Astronautical Sciences. 60(2): 303-312 (2013).

[12] Salvador, S., Chan, P.: FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space. 3rd Wkshp. on Mining Temporal and Sequential Data, ACM KDD '04. (2004)

[13] Srivastava, A., Klassen, E. P.: Functional and Shape Data Analysis. New York: Springer. (2016)

[14] Srivastava, A., Klassen, E. P., Joshi, S. H., Jermyn, I. H.: Shape Analysis of Elastic Curves in Euclidean Spaces. IEEE Trans. Pattern Analysis and Machine Intelligence. 33(7): 1415-1428 (2011).

[15] Umeyama, S.: Least-Squares Estimation of Transformation Parameters Between Two Point Patterns. IEEE Pattern Analysis & Machine Intelligence. 13(4): 376-380 (1991).

[16] Voronoi, G.: Nouvelles applications des paramètres continus à la théorie des formes quadratiques. J. Reine Angew. Math. 133: 97-178 (1908).

[17] Wahba, G.: A Least-Squares Estimate of Satellite Attitude. SIAM Review. 7(3): 409 (1965).