

# Fast Jacobi-type Algorithm for Computing Distances Between Linear Dynamical Systems

Nicolas D. Jimenez    Bijan Afsari    René Vidal  
Center for Imaging Science, Johns Hopkins University  
{njimenez, bijan, rvidal}@cis.jhu.edu

**Abstract**—The alignment distance is a novel metric between linear dynamical systems that has been shown to be very useful in many applications in computer vision. However, since the computation of the alignment distance requires solving a minimization problem on the orthogonal group, it is important to develop computationally efficient algorithms for solving this problem. In this paper, we present a fast and accurate Jacobi-type algorithm that solves this problem. Each step of the algorithm is equivalent to finding the roots of a quartic polynomial. We show that this rooting may be done efficiently and accurately using a careful implementation of Ferrari’s classical closed-form solution for quartic polynomials. For linear systems with orders that commonly arise in computer vision scenarios, our algorithm is roughly twenty times faster than a fast Riemannian gradient descent algorithm implementation and has comparable accuracy.

## I. INTRODUCTION

A distance between two linear dynamical systems (LDSs) is a useful notion and several attempts have been made to define distances between LDSs. In control theory such attempts go back to 1970’s (e.g., [12]) for applications in system identification. More recently, interest in distances between LDSs has resurfaced for applications in pattern recognition (e.g., classification and clustering) for high dimensional time-series data. Potential applications include the automatic classification of videos of human activities such as walking, running, jumping, and more complex human activities [5], as well as videos of natural dynamic scenes such as fountains, rivers, and smoke [7]. A common approach to this problem is to first estimate LDS parameters that model an observed video (or features extracted from the video), and then compare these parameters to a labelled training set of LDS parameters via an appropriate distance. In the 1-NN classification scheme, the observed video is assigned the label of the nearest video in the training set according to this distance (see, e.g., [4], [5] and references therein).

In [4], the *alignment* distance between LDSs was introduced. The main motivation for introducing the alignment distance was to overcome the following difficulty: every LDS admits an equivalence class of state-space realizations, related to each other via a linear change of basis in the state-space domain, which result in the same output sequence. It follows that two arbitrary parametrizations of LDSs cannot be compared directly. Hence, in the computation of the alignment distance one first solves an *alignment problem* to align the given realizations of two LDSs and then compares

the aligned realizations (see §II for more details). The alignment problem is a minimization problem over a *group of orthogonal matrices* for which a Riemannian gradient descent algorithm was introduced in [4].

In this paper, we introduce a fast and accurate Jacobi-type algorithm for computing alignment distances that results in considerable speed increases over Riemannian gradient descent. Our Jacobi-type algorithm solves the multidimensional optimization problem by solving a sequence of 1-dimensional subproblems. These 1-dimensional subproblems can be solved by prudent use of closed form solutions for quartic polynomials.

Besides the improvement in computation time, our algorithm is step-size free (a significant merit over the gradient descent algorithm), and, as our experiments show, is more likely than the gradient descent algorithm to find global minimizers. Although control applications are not the focus of this paper, we mention that the alignment distance could be potentially used in system identification or robust control applications.

This paper is organized as follows: In §II, we briefly review the methodology needed to define the alignment distance. In §III we describe our proposed algorithm for computing the alignment distance. In §IV numerical simulations are presented to validate the performance of our algorithms. Efficient MATLAB implementations of these algorithms may be downloaded from <http://www.vision.jhu.edu/code/>.

## II. BRIEF INTRODUCTION TO THE ALIGNMENT DISTANCE

We now briefly describe the theoretical basis for the alignment distance for *deterministic* LDSs. Our treatment is based on intuition, but it can be made rigorous via the machinery of differential geometry, specifically, the theory of fiber bundles.

Consider a discrete-time LDS  $M$  of order  $n$  and size  $(p, m)$  (i.e.,  $m$ -dimensional input and  $p$ -dimensional output) described by:

$$\begin{aligned} \mathbf{x}_t &= A\mathbf{x}_{t-1} + B\mathbf{v}_t \\ \mathbf{y}_t &= C\mathbf{x}_t, \end{aligned} \tag{1}$$

where  $R = (A, B, C) \in \tilde{\mathcal{L}}_{m,n,p} = \mathbb{R}^{n \times n} \times \mathbb{R}^{n \times m} \times \mathbb{R}^{p \times n}$  is a realization of  $M$ . Here,  $\mathbf{v}_t$  is the input which we assume to be deterministic. We call  $\tilde{\mathcal{L}}_{m,n,p}$  a realization space and we explicitly distinguish it from  $\mathcal{L}_{m,n,p}$  the space of LDSs

of size  $(p, m)$  and order  $n$ . The space  $\mathcal{L}_{m,n,p}$  is comprised of equivalent classes of realizations where  $(A, B, C)$  is equivalent to

$$P \circ (A, B, C) = (P^{-1}AP, B^{-1}P, CP), \quad (2)$$

for any  $P \in GL(n)$ , where  $GL(n)$  is the group of non-singular  $n \times n$  matrices.

Due to this equivalence relation, we cannot compare two systems  $M_1$  and  $M_1$  by simply comparing their arbitrary realizations  $R_1$  and  $R_2$ . Instead, one needs first to optimally align the two realizations (i.e., by finding a transformation  $P$  that brings the  $R_1$  and  $R_2$  as close to each other as possible). However, due to certain theoretical and computational difficulties, mostly stemming from *non-compactness* of  $GL(n)$ , such an alignment is difficult to compute. To address this issue, we propose to standardize the realizations before aligning them. By standardization, intuitively, we mean a step in which the non-compact part of  $GL(n)$  is thrown out and only its maximal compact subgroup  $O(n)$ , the group of  $n \times n$  orthogonal matrices, is kept. It can be shown that standardization is possible for a large class of LDSs. However, here, we only consider it in relation to a specific class of LDSs which we call tall and full rank. This class appears naturally as the output of a system identification algorithm popular in video sequence analysis (see [4], [7]).

More specifically, let  $\mathcal{L}_{m,n,p}^{\text{IC}} = \{R \in \tilde{\mathcal{L}}_{m,n,p} : \text{rank}(C) = n\}$ , i.e., realizations in which  $C$  is of full column rank. Denote the corresponding space of LDSs by  $\mathcal{L}_{m,n,p}^{\text{IC}}$ . A simple standardized space related to  $\mathcal{L}_{m,n,p}^{\text{IC}}$  is  $\widetilde{\mathcal{O}}\mathcal{L}_{m,n,p}^{\text{IC}} = \{R \in \tilde{\mathcal{L}}_{m,n,p}^{\text{IC}} : C^T C = I_n\}$ , where  $I_n$  is the  $n \times n$  identity matrix and  $^T$  denotes matrix transpose. Notice that any realization in  $\mathcal{L}_{m,n,p}^{\text{IC}}$  can be standardized, e.g. via the SVD of  $C$ . Moreover, on  $\widetilde{\mathcal{O}}\mathcal{L}_{m,n,p}^{\text{IC}}$ ,  $GL(n)$  acts only through its subgroup  $O(n)$ , since  $P \circ R \in \widetilde{\mathcal{O}}\mathcal{L}_{m,n,p}^{\text{IC}}$  implies  $P \in O(n)$  for any  $P \in GL(n)$  and  $R \in \widetilde{\mathcal{O}}\mathcal{L}_{m,n,p}^{\text{IC}}$ .

Upon standardization, aligning the realizations and comparison of LDSs become straightforward. More specifically, given any realizations  $R_1, R_2 \in \widetilde{\mathcal{O}}\mathcal{L}_{m,n,p}^{\text{IC}}$  of LDSs  $M_1, M_2 \in \mathcal{L}_{m,n,p}^{\text{IC}}$ , we define the distance

$$d^2(M_1, M_2) = \min_{Q \in O(n)} d_F^2(Q \circ R_1, R_2), \quad (3)$$

where  $d_F$  is the following Frobenius norm based distance on  $\widetilde{\mathcal{O}}\mathcal{L}_{m,n,p}^{\text{IC}}$ :

$$d_F^2(R_1, R_2) = \lambda_A \|A_1 - A_2\|_F^2 + \lambda_B \|B_1 - B_2\|_F^2 + \lambda_C \|C_1 - C_2\|_F^2, \quad (4)$$

The minimization in (3) is called the (*realization*) *alignment* problem and can be explicitly expressed as

$$d^2(M_1, M_2) = \min_{Q \in O(n)} f(Q; R_1, R_2), \quad (5)$$

where

$$f(Q; R_1, R_2) = \lambda_A \|Q^T A_1 Q - A_2\|_F^2 + \lambda_B \|Q^T B_1 - B_2\|_F^2 + \lambda_C \|C_1 Q - C_2\|_F^2. \quad (6)$$

Note that the parameters  $\lambda_A, \lambda_B, \lambda_C$  determine the relative contributions of the  $A, B, C$  matrices to the cost function. As such, they allow the possibility of tuning the metric for a particular application.

### III. JACOBI-TYPE ALGORITHM FOR COMPUTING THE ALIGNMENT DISTANCE

We now consider the minimization of Equation (6) via a Jacobi type algorithm. The Jacobi eigenvalue algorithm is a well established method in numerical linear algebra [10]. More generally, Jacobi-type methods have been used for minimizing functions defined on  $O(n)$  (e.g., joint diagonalization). Such methods are essentially coordinate descent optimizations on the manifold  $O(n)$ , i.e., one performs a sequence of 1-dimensional minimizations along fixed directions. These directions, in fact, point along the components of an orthogonal basis of the tangent space at each point in  $O(n)$ . The amount of descent is preferably determined by finding the exact solution to the 1-dimensional minimization sub-problem. Due to the specific selection of the descent directions, the 1-dimensional sub-problems are usually easy to solve, and in fact a *global* minimizer can often be found. A major benefit of Jacobi-type methods is their local quadratic rate of convergence [11]. These methods can be extended to groups other than  $O(n)$ , including non-compact groups (see, e.g. [3], [11], [13]).

#### A. Jacobi-type Algorithm and Lagrangian for Minimizing $f$

Consider the optimization problem in (6), where our goal is to minimize

$$f(Q; R_1, R_2) = d_F^2(Q \circ R_1, R_2) \quad (7)$$

over  $Q \in O(n)$ . Recall that  $O(n)$  has two connected components:  $SO(n)$ , which is comprised of orthogonal matrices of determinant  $+1$ , and  $O^-(n)$ , whose elements have determinant  $-1$ . Thus, to minimize  $f(Q; R_1, R_2)$  in (6), we choose initial values for  $Q$ , with at least one initial value in  $SO(n)$  and one in  $O^-(n)$ , and update  $R_1$  as follows:  $R_1 \leftarrow Q \circ R_1$ . Given this initialization, we perform updates of the form

$$Q \leftarrow Q Q_{pq}(\theta_{pq}) \quad \text{and} \quad R_1 \leftarrow Q_{pq}(\theta_{pq}) \circ R_1, \quad (8)$$

where  $Q_{pq}(\theta)$  is the Givens rotation matrix

$$Q_{pq}(\theta) = \begin{bmatrix} 1 & \cdots & p & \cdots & q & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ p \rightarrow & \cdots & c & \cdots & -s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ q \rightarrow & \cdots & s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \quad (9)$$

with  $c = \cos(\theta)$ ,  $s = \sin(\theta)$ , and  $\theta \in (-\pi, \pi]$ . The algorithm usually begins from  $(p, q) = (1, 2)$  and progresses by sweeping through all index pairs  $(p, q)$  such that  $q > p$  (this is called one *sweep*) and then repeats the process till

convergence. It is a well known fact from linear algebra that every orthogonal matrix can be represented by a product of Givens rotation matrices. Hence, Jacobi-type algorithms such as ours are able to search the entire space of orthogonal matrices to minimize the function in question.

To find  $\theta_{pq}$  we need to solve the following one-dimensional optimization problem:  $\min_{\theta} f(Q_{pq}(\theta); R_1, R_2)$ . However, as we shall see it is convenient to do the minimization over  $s, c$  subject to the constraint  $s^2 + c^2 = 1$ . We can then write the minimization problem as  $\min_{c^2+s^2=1} f(Q_{pq}(c, s); R_1, R_2)$ . It can be shown by simple algebraic manipulation that this minimization is equivalent to maximizing

$$g(Q_{pq}(c, s); R_1, R_2) = \lambda_A \text{tr}(Q^T A_1^T Q A_2) + \text{tr}(Q^T D), \quad (10)$$

subject to the constraint  $c^2 + s^2 = 1$ , where  $D = \lambda_B(B_1 B_2^T) + \lambda_C(C_1^T C_2)$ . We denote the elements of the defined matrices as follows:  $[A_1]_{ij} = a_{ij}$ ,  $[A_2]_{ij} = a'_{ij}$ ,  $[D]_{ij} = d_{ij}$ . A straightforward computation shows that

$$g(Q_{pq}(c, s); R_1, R_2) = k_0 c^2 + k_1 s^2 + k_2 c s + k_3 c + k_4 s, \quad (11)$$

where

$$\begin{aligned} k_0 &= \lambda_A [a_{pp} a'_{pp} + a_{pq} a'_{pq} + a_{qq} a'_{qq} + a_{qp} a'_{qp}] \\ k_1 &= \lambda_A [a_{pp} a'_{qq} + a_{qq} a'_{pp} - a_{pq} a'_{qp} - a_{qp} a'_{pq}] \\ k_2 &= \lambda_A [a_{pp} (-a'_{qp} - a'_{pq}) + a_{pq} (a'_{pp} - a'_{qq}) + \\ &\quad a_{qp} (a'_{pp} - a'_{qq}) + a_{qq} (a'_{pq} + a'_{qp})] \\ k_3 &= d_{pp} + d_{qq} + \lambda_A \sum_{j \neq p, q} a_{jp} a'_{jp} + a_{jq} a'_{jq} + a_{pj} a'_{pj} + a_{qj} a'_{qj} \\ k_4 &= d_{qp} - d_{pq} + \lambda_A \sum_{j \neq p, q} a_{jq} a'_{jp} + a_{qj} a'_{pj} - a_{jp} a'_{jq} - a_{pj} a'_{qj} \end{aligned} \quad (12)$$

To maximize (11) we form the Lagrangian under the constraint  $c^2 + s^2 = 1$ :

$$L(c, s, \lambda) = k_0 c^2 + k_1 s^2 + k_2 c s + k_3 c + k_4 s - \lambda (c^2 + s^2 - 1). \quad (13)$$

The first order optimality conditions yield:

$$\begin{aligned} 2k_0 c + k_2 s + k_3 &= 2\lambda c \\ 2k_1 s + k_2 c + k_4 &= 2\lambda s \\ c^2 + s^2 &= 1, \end{aligned} \quad (14)$$

Solving this set of equations for  $c$  we obtain:

$$P_c(c) = \alpha_4 c^4 + \alpha_3 c^3 + \alpha_2 c^2 + \alpha_1 c + \alpha_0 = 0, \quad (15)$$

where

$$\begin{aligned} \alpha_4 &= 4k_2^2 + 4(k_0 - k_1)^2, \quad \alpha_3 = 4k_3(k_0 - k_1) + 4k_2 k_4, \\ \alpha_0 &= k_2^2 - k_3^2, \quad \alpha_1 = 4k_3(k_1 - k_0) - 2k_2 k_4, \\ \alpha_2 &= k_4^2 - 4k_2^2 + k_3^2 - 4(k_0 - k_1)^2. \end{aligned} \quad (16)$$

Alternatively, we may solve for  $s$ , obtaining

$$P_s(s) = \beta_4 s^4 + \beta_3 s^3 + \beta_2 s^2 + \beta_1 s + \beta_0 = 0, \quad (17)$$

where

$$\begin{aligned} \beta_4 &= 4k_2^2 + 4(k_1 - k_0)^2, \quad \beta_3 = 4k_4(k_1 - k_0) + 4k_2 k_3, \\ \beta_0 &= k_2^2 - k_4^2, \quad \beta_1 = 4k_4(k_0 - k_1) - 2k_2 k_3, \\ \beta_2 &= k_3^2 - 4k_2^2 + k_4^2 - 4(k_1 - k_0)^2. \end{aligned} \quad (18)$$

The relation between the roots of these two polynomials is as follows:

$$s = \frac{2k_2 c^2 + k_4 c - k_2}{(2k_0 - 2k_1)c + k_3}, \quad c = \frac{2k_2 s^2 + k_3 s - k_2}{(2k_1 - 2k_0)s + k_4}. \quad (19)$$

To find the optimal  $(c, s)$  pairs that maximize (11), we must root (15) or (17) and then make a substitution using (19) to obtain the  $(c, s)$  pair. Unfortunately, (19) was derived supposing that the  $(c, s)$  pair is a critical point of (11). If the computed roots are inexact, the evaluation of (19) will result in  $c^2 + s^2 \neq 1$ . Therefore, care must be taken to ensure that inaccuracies in the rooting of these polynomials do not result in non-orthogonal updates. We adopt the following method to circumvent this issue. Suppose that we have obtained a root  $c$ . To compute the corresponding  $s$  while ensuring the orthogonality of the matrix  $Q_{pq}(\theta)$ , we find:

$$s = \text{sign}(s') \sqrt{1 - c^2}, \quad \text{where } s' = \frac{2k_2 c^2 + k_4 c - k_2}{(2k_0 - 2k_1)c + k_3}. \quad (20)$$

The next step is to evaluate the cost function for each obtained  $(c, s)$  pair. The  $(c, s)$  pair that results in the largest value of (11) is then used to perform the Jacobi update. Since the matrices  $Q_{pq}(c, s)$  are sparse, these updates can be executed efficiently using standard Jacobi-update algorithms such as those found in [10].

### B. Accurate and Fast Quartic Root Finding

Obtaining the solutions to Equation (15) or (17) in a fast and accurate manner is crucial to ensure the overall speed and accuracy of our algorithm. Interestingly, optimizing functions of the form (10) have appeared in the literature [6], most notably in the orthogonal joint diagonalization problem. To the best of our knowledge, the related fast and accurate rooting has not been addressed in the literature.

Our approach combines a closed form solution method with a Newton-Raphson root polishing step to obtain the roots of Equation (15) or (17). The first closed form solution method for quartics was derived by Ferrari in 1540 and is described in [1]. One benefit of such a formula is that it yields all four roots of the polynomial and hence we can, in principle, find global minimizers for our 1-dimensional minimization problems. Another benefit is in terms of speed: a straightforward C implementation of Ferrari's method is about 17 times faster than MATLAB's quasi-standard polynomial rooting function, which performs the QR algorithm on a polynomial's companion matrix. Nevertheless Ferrari's method is not popular [14] since it is highly prone to round off error. Strobach recently proposed a fast and accurate (general purpose) quartic root finding method based on iterative refinement of Ferrari's solution [14]. Strobach's method is a viable option in our problem. However, in our application we can achieve the same accuracy as Strobach's method in better

---

**Algorithm 1:** Jacobi-type algorithm for computing  $d^2(M_1, M_2)$ 


---

**Data:** Realizations  $R_1 = (A_{10}, B_{10}, C_{10}) \in \widetilde{\mathcal{O}}_{\Sigma_{m,n,p}}$  and  $R_2 = (A_{20}, B_{20}, C_{20}) \in \mathcal{O}_{\Sigma_{m,n,p}}$  of LDSs  $M_1$  and  $M_2$  in  $\Sigma_{m,n,p}$ , weight parameters  $\lambda_A, \lambda_B, \lambda_C$ , a set of  $2l$  orthogonal matrices to be used for algorithm initializations  $\{Q_+^1, \dots, Q_+^l, Q_-^{l+1}, \dots, Q_-^{2l}\}$ , where  $Q_+^i \in SO(n)$  and  $Q_-^j \in O^-(n)$ , and stopping criteria  $\text{StopCrit} \geq 0$ .

**Result:**  $Q \in O(n)$  that minimizes  $f(Q; R_1, R_2)$  and  $d^2 = f(Q; R_1, R_2)$ .

```

for  $j = 1 : 2l$  do
  StopFlag  $\leftarrow 0$ ;
   $Q \leftarrow Q_{\pm}^j$ ;
   $A_1 \leftarrow Q^\top A_{10} Q$ ;
   $D \leftarrow Q^\top (\lambda_B (B_{10} B_{20}^\top) + \lambda_C (C_{10}^\top C_{20}))$ ;
   $i = 1$ ;
  while StopFlag = 0 do
    StopFlag  $\leftarrow 1$ ;
    for  $p = 1 : n - 1$  do
      for  $q = p + 1 : n$  do
         $\mathbf{k} \leftarrow \text{CalcK}(A_1, A_2, D, \lambda_A)$ ; // Eq (12)
         $\alpha \leftarrow \text{CalcAlpha}(\mathbf{k})$ ; // Eq (16)
         $[c, s] = \text{JacobiAngle}(\alpha, \mathbf{k})$ ; // Alg (2)
        if  $\text{abs}(s) \leq \text{StopCrit}$  then
          StopFlag  $\leftarrow 0$ ;
           $A_1 \leftarrow Q_{pq}^\top A_1 Q_{pq}$ ;
           $D \leftarrow Q_{pq}^\top D$ ;
           $Q \leftarrow Q Q_{pq}$ ;
      end for
    end for
  end while
 $d^2(M_1, M_2) = f(Q; R_1, R_2)$ ; // Eq (6)

```

---

time. Note that since we are only interested in roots in  $[-1, 1]$  our problem is easier than general rooting (as Strobach's method). We simply use the roots returned by Ferrari's algorithm to initialize a Newton-Raphson root-polishing iteration. If the estimates of the roots are sufficiently accurate, a few iterations of Newton-Raphson improve their accuracy considerably as the number of significant digits nearly doubles at each iteration [9].

Another question is whether to solve for  $\cos(\theta)$  or for  $\sin(\theta)$ . Strobach observed [14] that Ferrari's algorithm performs poorly when there is a large *root spread*, where this quantity is defined as the ratio of the maximal magnitude of a root to that of the minimal magnitude:  $S = \frac{|r_{\max}|}{|r_{\min}|}$ . As the algorithm converges  $\theta_{pq} \rightarrow 0$  and hence at least one of the roots of the  $P_s(s)$  polynomial approaches zero. In this case, the root spread becomes arbitrarily large. By solving the  $P_c(c)$  polynomial, we avoid this situation.

We have observed that Ferrari's method in combination with Newton-Raphson is about 3 times faster than Strobach's procedure, is easier to implement as it involves no parallel programming, and has comparable accuracy. Our method of computing Jacobi angles for our optimization sub-problem is summarized in Algorithm 2. In §IV, we present numerical evidence which confirms our claim that a Ferrari-based optimization which uses  $\cos(\theta)$  is preferable (e.g., see Figure

---

**Algorithm 2:** Jacobi angle computation

---

**Data:** Coefficient vector  $\mathbf{k} = [k_0, k_1, k_2, k_3, k_4]$  that defines  $g(\cos(\theta), \sin(\theta), \mathbf{k})$  as in Eq (11), and polynomial coefficient vector  $\alpha = [\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4]$ .

**Result:** The optimal  $(c, s)$  Jacobi update.

```

 $\mathbf{c} \leftarrow \text{FerrariRoots}(\alpha)$ ;
 $\mathbf{c} \leftarrow \text{NewtonRaphsonPolish}(\mathbf{c})$ ;
 $\mathbf{s} \leftarrow \text{CalcSine}(\mathbf{c}, \mathbf{k})$ ; // Eq (20)
 $\text{cost}[0] \leftarrow g(c = 1, s = 0, \mathbf{k})$ ; // Eq (11)
for  $i = 1 : \text{size}(\mathbf{c})$ ; // find  $(c, s)$  pair with
largest cost
do
   $\text{cost}[i] = g(\mathbf{c}(i), \mathbf{s}(i), \mathbf{k})$ ;
end
 $\text{ind} \leftarrow \text{MaxInd}(\text{cost})$ ;
if  $\text{ind} = 1$  then
  break; // Skip  $(p, q)$  pair in Alg (1)
else
   $c = \mathbf{c}(\text{ind})$ ;
   $s = \mathbf{s}(\text{ind})$ 
end

```

---

1).

### C. Handling Local Minima

Unfortunately,  $f$  is not convex on  $O(n)$  and it may have local minimizers that are not global. In fact, it is known that the only continuous convex function on a compact manifold is the constant function. Although we find global minimizers for the 1-dimensional sub-problems there is no guarantee that the algorithm will converge to the global minimizer of the main problem. A simple method to mitigate this difficulty is to begin the optimization with different initial matrices  $Q$ . For example, Algorithm 1 may be repeated with several random (uniformly distributed) initial points. Alternatively, the algorithm maybe repeated for a set of initial matrices  $Q_+^l = \{Q^1, \dots, Q^l\} \subset SO(n)$  and  $Q_-^l = \{Q^{l+1}, \dots, Q^{2l}\} \subset O(n)$ , where the matrices in each set are “maximally separated” in some sense. In fact, if such a set is available by simple function evaluations one may guess a good initial point and avoid running the algorithm  $2l$  times. One simple strategy to find the set  $Q_+^l$  is by maximizing an energy function such as

$$E(\{Q_i\}) = \log \prod_{i \neq j} \|Q_i - Q_j\|_F^2 = \sum_{i \neq j} \log \|Q_i - Q_j\|_F^2. \quad (21)$$

Notice that a set  $Q_-^l$  can be found simply by swapping the first and second columns of each matrix in  $Q_+^l$ . This energy function is known as the logarithmic energy [8]. We can use a gradient method to find these matrices off-line (although this is not a completely satisfactory solution, since again we have no guarantee to find the global maximizers.) For  $l = 2$  a solution is simply  $Q_+^2 = \{I_n, -I_n\}$  if  $n$  is even and if  $n$  is odd  $Q_+^2 = \{I_n, I_n^-\}$ , where  $I_n^- = \text{diag}(-1, \dots, -1, +1)$ .

### D. Comparison with Riemannian gradient descent

A Riemannian gradient descent algorithm for minimizing  $f(Q; R_1, R_2)$  is presented in [4]. Each step of standard

Riemannian gradient descent on  $SO(n)$  involves computing the matrix exponential. Instead, one could use the polar decomposition (e.g., via the SVD) or the QR factorization (such maps are called retractions [2]) to reduce the computational cost of computing the matrix exponential. Both these approaches are of order  $\mathcal{O}(n^3)$  (our experiments show that, as expected, the SVD-based method is several times slower but much more accurate). Moreover, to ensure decrease of the cost at each iteration one needs to implement a step-size rule such as Armijo's rule [2] which involves function evaluations and adds to the computational cost. On the other hand, a single sweep of our Jacobi-type algorithm requires  $\mathcal{O}(n^2)$  updates. For each index pair, the computation of the coefficients in (12) and the Jacobi pair  $(c, s)$  are of order  $\mathcal{O}(n)$ . Hence, the overall complexity of each step of our proposed Jacobi algorithm is  $\mathcal{O}(n^3)$ , which is the same complexity as Riemannian gradient descent. However, Jacobi-type methods have quadratic local convergence whereas the convergence for gradient descent is linear.

#### IV. NUMERICAL EXPERIMENTS

In this section, we present numerical experiments that show the efficiency and accuracy of our algorithm. We perform our experiments on systems generated at random. More precisely, we generate random realizations on  $\mathcal{L}_{m,n,p}^{\text{C,a}}$ , the asymptotically stable subset of  $\mathcal{L}_{m,n,p}^{\text{C}}$  (although the stability condition is not necessary as far as our computations are concerned). Each random realization will induce a random system. We choose  $m = n = 5$  and  $p = 10$ . The space  $\mathcal{L}_{m,n,p}^{\text{C,a}}$  with large  $p$  and  $n \approx 5$  is the space that naturally appears in the analysis of video sequence data (see [4]).

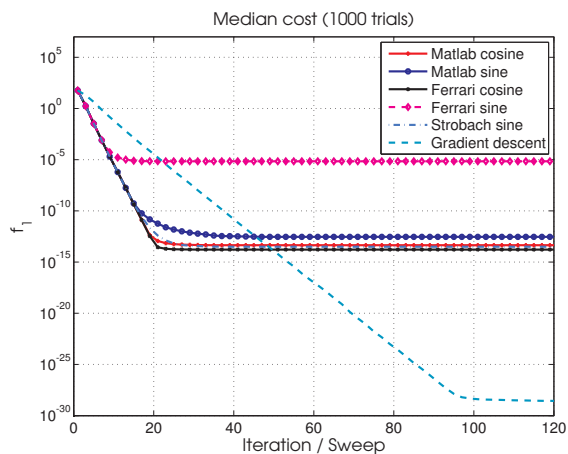
**Data generation and algorithm settings.** To generate two random systems  $M_1, M_2 \in \mathcal{L}_{5,5,10}^{\text{C,a}}$ , where  $M_2$  is a perturbation of  $M_1$ , we generate random realizations  $R_1, R_2 \in \mathcal{O}\mathcal{L}_{5,5,10}^{\text{C,a}}$ , ( $R_i = (A_i, B_i, C_i)$ ) as follows. We first generate random matrices  $S_1, \dots, S_5 \in \mathbb{R}^{5 \times 5}$  and  $T_1, T_2 \in \mathbb{R}^{10 \times 5}$  with unit Gaussian elements. To generate a random stable matrix  $A_1 \in \mathbb{R}^{5 \times 5}$ , we compute the matrix exponential  $A_1 = \exp(S_1 - S_1^\top - S_2 S_2^\top)$ . This ensures that the magnitude of the eigenvalues of  $A_1$  is less than 1. We set  $B_1 = S_3$ . To generate a random orthogonal matrix  $C_1$ , we compute the singular value decomposition (SVD)  $T_1 = U \Sigma V^\top$  and set  $C_1 = U$ . To generate realization  $R_2$  from  $R_1$ , we first generate a random orthogonal matrix  $Q_i \in O(n)$ . We then set  $A_2 = Q_i^\top (A_1 + \sigma S_4) Q_i$  and  $B_2 = Q_i^\top (B_1 + \sigma S_5)$ . Likewise, to generate  $C_2$ , we set  $C_2' = (C_1 + \sigma T_2) Q_i$ . Since this matrix is not orthogonal, we compute the SVD,  $C_2' = U \Sigma V^\top$  and set  $C_2 = U$ . Notice that if  $\sigma = 0$ , then  $R_2 = Q_i \circ R_1$  (see (2)) and hence  $M_1 = M_2$ . Now we generate  $T = 1000$  of such  $(M_1, M_2)$  pairs with  $\sigma \in \{0, 0.2\}$ , and run our Jacobi algorithm to find the distance  $d_1(M_1, M_2)$  (with  $\lambda_A, \lambda_B, \lambda_C = 1$ ). We use initial sets  $\mathcal{Q}_+^2$  and  $\mathcal{Q}_-^2$  as in §III-C. We implement our Jacobi algorithm using MATLAB functions as well as an optimized MATLAB executable file (MEX file). Since our implementation of gradient descent also uses a combination of MATLAB and MEX functions, we consider our comparison to be fair.

**Accuracy Comparison.** The results for  $\sigma = 0$  are shown in Figure 1, which shows the point-wise median of the cost function (across  $T = 1000$  samples) in terms of the number of iterations (or sweeps) of the algorithms. We use the median simply to show a typical behavior. The results for both the Jacobi-type algorithm and the gradient descent (with SVD and Armijo step-size §III-D) are shown. Moreover, several possibilities for root-finding are examined, these include: `Strobach+cos`, `Strobach+sin`, `Matlab+cos`, `Matlab+sin`, `Ferrari+cos`, and `Ferrari+sin`. The Ferrari methods use at most 5 iterations of Newton-Raphson root polishing per root. `Strobach+cos` was not shown in this figure, as its performance was indistinguishable from Ferrari cosine. Since our implementation of Ferrari's method in Algorithm 2 is roughly three times faster than Strobach's general purpose quartic solver, we conclude that `Ferrari+cos` is the best among these root solving method for our Jacobi type algorithm. Figure 1 also shows that the difference in accuracy between `Ferrari+cos` and `Ferrari+sin` is significant. This difference in accuracy is much more pronounced for the Ferrari solver than for the Matlab and Strobach solvers, where the difference between using  $\cos(\theta)$  and  $\sin(\theta)$  is negligible (this justifies our choice of solving (15) instead of (17)).

Note that the Jacobi and gradient descent algorithms are both limited in the accuracy they may attain, since the minimum angles of rotation are limited by the machine epsilon. It appears that this limitation affects our Jacobi algorithm before affecting gradient descent. As both algorithms converge, the Jacobi algorithm will need to take smaller steps than gradient descent, as it is constrained to move in fixed directions and cannot move directly towards the minimum. As a consequence, it is expected that machine epsilon affects the Jacobi algorithm before affecting gradient descent. This explains the difference in maximal accuracy between the two methods.

**Speed comparison.** Figure 1 clearly shows superiority of our Jacobi-type algorithm with respect to gradient descent in terms of the number of iterations. Another speed comparison for the more realistic  $\sigma = 0.2$  scenario is given in Table I and in Figure 2, which shows the cumulative percentage of runs that converged in terms of number of iterations (sweeps). We consider that the algorithm converged whenever the algorithm reaches within  $10^{-4}$  of the lowest value of  $d^2(M_1, M_2)$  computed over 300 sweeps/iterations. Time per 1000 iterations / sweeps was measured on a HP Pavilion G7 laptop. The second column in Table I measures the average number of iterations needed to converge to  $10^{-4}$  of the lowest cost for a given computation of  $d^2(M_1, M_2)$ . The third column is obtained by multiplying the first two. These results demonstrate that, in this specific example, our proposed Jacobi method converges roughly 20 times faster than Riemannian gradient descent.

**Local minima.** Next, we examine how algorithms perform when initialized only with one initial condition, namely,  $\mathcal{Q}_+^1 = \{I_5\}$  and  $\mathcal{Q}_-^1 = \{-I_5\}$ . When  $\sigma = 0$  we consider



**Fig. 1** – Typical convergence behavior for different algorithms and root finding methods ( $\sigma = 0$ ).

	Jacobi	Gradient descent
Time per 10,000 sweeps / iterations	0.50 s	4.54 s
Mean iterations to converge	19.2	43.2
Mean time to converge	1x	20.43x

**TABLE I** – Speed comparison between gradient descent and our proposed Jacobi method for the  $\sigma = 0.2$  case.

that an algorithm “converged to a global solution,” if the final computed value is less than  $10^{-4}$ . In Table II, we see that using  $Q_+^2$  and  $Q_-^2$  improves global convergence when  $\sigma = 0$  for the gradient descent. However, interestingly, our Jacobi algorithm already avoids local minima with  $l = 1$  and seems to be less propense to local minima than the gradient descent. This might be attributed to the fact that our Jacobi algorithm actually solves the 1-dimensional sub-problems globally.

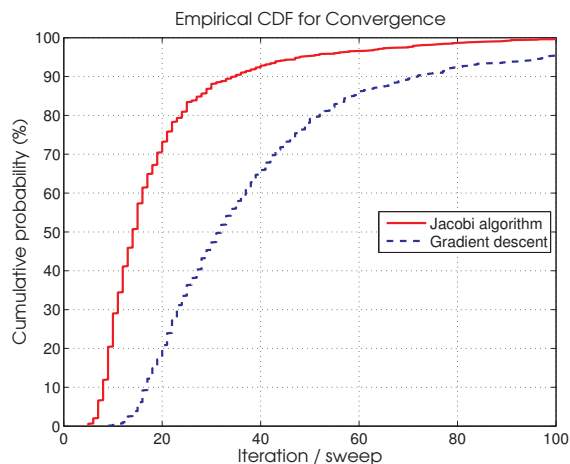
## V. DISCUSSION

We have proposed a fast and accurate Jacobi-type algorithm for computing alignment distances between LDSs. Our algorithm is significantly faster than gradient descent for linear systems with orders that commonly arise in computer vision scenarios, and has comparable accuracy. Moreover, we hope that our insights will be useful in the investigation of other optimization problems involving orthogonality constraints.

**Acknowledgements.** This work has been supported in part by the grants ONR N00014-09-10084, NSF CAREER 0447739, NSF 0941362, NSF 0941463 and NSF 0931805.

## REFERENCES

- [1] M. Abramowitz and I. Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, volume 55. Dover publications, 1964.
- [2] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ, 2008.
- [3] B. Afsari. Simple LU and QR based non-orthogonal matrix joint diagonalization. In J. P. Rosca, D. Erdogmus, J. C. P., and S. Haykin, editors, *ICA*, volume 3889 of *Lecture Notes in Computer Science*, pages 1–7. Springer, 2006.



**Fig. 2** – Empirical cumulative probability of convergence in terms of number of iterations ( $\sigma = 0.2$ ).

Number of initializations $l$	1	2
Jacobi global convergence (%)	100.0	100.0
Gradient global convergence (%)	96.5	98.0

**TABLE II** – Convergence of gradient descent to global minimizer(s) is enhanced by using two initializations ( $\sigma = 0$ ).

- [4] B. Afsari, R. Chaudhry, A. Ravichandran, and R. Vidal. Group action induced distances for averaging and clustering linear dynamical systems with applications to the analysis of dynamic scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Providence, RI, 2012.
- [5] B. Béjar, L. Zappella, and R. Vidal. Surgical gesture classification from video data. In N. Ayache, H. Delingette, P. Golland, and K. Mori, editors, *Medical Image Computing and Computer-Assisted Intervention*, pages 34–41, 2012.
- [6] J.-F. Cardoso and A. Souloumiac. Jacobi angles for simultaneous diagonalization. *SIAM J. Mat. Anal. Appl.*, 17(1):161–164, January 1996.
- [7] G. Doretto, A. Chiuso, Y. Wu, and S. Soatto. Dynamic textures. *International Journal of Computer Vision*, 51(2):91–109, 2003.
- [8] P. Draznev, D. Legg, and D. Townsend. Discrete logarithmic energy on the sphere. *Pacific J. Math*, 207(2):345–358, 2002.
- [9] B. Flannery, W. Press, S. Teukolsky, and W. Vetterling. Numerical recipes in C. *Press Syndicate of the University of Cambridge, New York*, 1992.
- [10] G. Golub and c. F. Van Loan. *Matrix computations*, volume 3. Johns Hopkins Univ Press, 1996.
- [11] U. Helmke and K. Hüper. A Jacobi-type method for computing balanced realizations. *Systems & Control letters*, 39(19-30), 2000.
- [12] P. S. Krishnaprasad. *Geometry of Minimal Systems and the Identification Problem*. PhD thesis, Harvard University, 1977.
- [13] A. Souloumiac. Nonorthogonal joint diagonalization by combining Givens and hyperbolic rotations. *IEEE Transactions on Signal Processing*, 57:2222–2231, June 2009.
- [14] P. Strobach. The fast quartic solver. *Journal of computational and applied mathematics*, 234(10):3007–3024, 2010.