

Geometrical Methods for Non-negative ICA: Manifolds, Lie Groups and Toral Subalgebras

Mark D. Plumbley

*Department of Electronic Engineering, Queen Mary University of London, Mile
End Road, London E1 4NS, UK*

Abstract

We explore the use of geometrical methods to tackle the non-negative independent component analysis (non-negative ICA) problem, without assuming the reader has an existing background in differential geometry. We concentrate on methods that achieve this by minimizing a cost function over the space of orthogonal matrices. We introduce the idea of the manifold and Lie group $SO(n)$ of special orthogonal matrices that we wish to search over, and explain how this is related to the Lie algebra $\mathfrak{so}(n)$ of skew-symmetric matrices. We describe how familiar optimization methods such as steepest-descent and conjugate gradients can be transformed into this Lie group setting, and how the Newton update step has an alternative Fourier version in $SO(n)$. Finally we introduce the concept of a toral subgroup generated by a particular element of the Lie group or Lie algebra, and explore how this commutative subgroup might be used to simplify searches on our constraint surface. No proofs are presented in this article.

Key words: Independent component analysis, Nonnegative ICA, Lie group, orthogonal rotations, toral subgroup, toral subalgebra

1 Introduction

The last few years have seen a great increase in interest in the use of geometrical methods for independent component analysis (ICA), as evidenced for example by the current special issue. Some of this research involves some rather difficult-sounding concepts such as Stiefel manifolds, Lie groups, tangent planes and so on that can make this work rather hard going for a reader with a more traditional ICA or neural networks background. However, I believe that the concepts underlying these methods are so important that they cannot be left to a few ICA researchers with a mathematics or mathematical physics background to investigate, while others are content to work on applications using “ordinary” methods. The aim of the current paper is therefore to explore some of these geometrical methods, using the example of the *non-negative ICA* task, while keeping the more inaccessible concepts to a minimum. As we explore these geometrical approaches, we may have to revise our intuition about some things that we often take for granted, such as the concept of *gradient*, or the ubiquitous method of *steepest descent*.

We already work with geometrical methods of one sort or another, and if we are already working with ICA we probably have pictures in our minds of what is happening when we perform a steepest descent search, or a line search with a Newton step, or maybe even a conjugate gradients method. But we are usually used to working in large, flat, even spaces with a simple and well-defined Euclidean distance measure. Moving around our space is easy and *commutative*: $\Delta x + \Delta y = \Delta y + \Delta x$, so we can make our movements in any order we like, and end up at the same place (i.e. our movements *commute*)

Email address: mark.plumbley@elec.qmul.ac.uk (Mark D. Plumbley).

But in the current article we will be exploring some geometry over curved spaces, or *manifolds*, where some or all of our cherished assumptions, even assumptions we may not realize we were making, will no longer work. This doesn't mean they are difficult *per se*, only that we have to be a little bit more careful until we get used to them.

There are no “proofs” in this article. I apologize in advance to the more mathematical reader for the liberties I have taken with notation and rigour. If you are already familiar with the type of Lie group methods discussed here, I hope that you will make allowances for the unusual approach taken here in the name of accessibility.

The article is organized as follows: First we introduce the non-negative ICA task, consider approaches based on steepest descent, and discuss imposition of orthonormality constraints using projection and tangent methods. We then describe the concept of a *Lie group*, with its corresponding *Lie algebra*, and explore the group of orthogonal matrices, in particular the group of special orthogonal matrices $SO(n)$. We consider optimizing over $SO(n)$, using rotation for $n = 2$, geodesic flow and geodesic search for more general n , and conjugate gradient methods. We introduce the concept of a *toral subgroup* and link this to the issue of matrix exponentiation. Finally we briefly discuss other related methods before concluding.

2 The non-negative ICA task

We consider the task of independent component analysis (ICA), in its simplest form, to be that of estimating a sequence of source vectors $\mathbf{s} = (s_1, \dots, s_n)$

and an $n \times n$ mixing matrix \mathbf{A} given only the observations of a sequence of mixtures $\mathbf{x} = (x_1, \dots, x_n)$. These are assumed to be linearly generated by

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{1}$$

where we assume that any noise in the observations \mathbf{x} can be neglected. There is a *scaling ambiguity* between the sources \mathbf{s} and the mixing matrix \mathbf{A} , meaning that we can scale up any source s_i and scale down any column vector \mathbf{a}_i and have the same observation \mathbf{x} , so we conventionally assume that the sources have unit variance. For the *non-negative ICA* task, which we concentrate on here, we also assume that the sources \mathbf{s} must always be non-negative, i.e. there is a zero probability that any source is negative: $\Pr(s_i < 0) = 0$. (For technical reasons, we also assume that the sources are *well grounded*, $\Pr(s_i < \delta) > 0$ for any $\delta > 0$, i.e. that the probability distribution of all of the sources “goes all the way down to zero” [1]). In common with many other ICA methods, non-negative ICA can be tackled in two stages: *whitening* followed by *rotation*.

Whitening In the first stage, the input data \mathbf{x} is *whitened* to give a new sequence of vectors $\mathbf{z} = \mathbf{V}\mathbf{x}$, where \mathbf{V} is chosen such that each element of the vector sequence $\mathbf{z} = (z_1, \dots, z_n)$ has unit variance and all are uncorrelated from each other, i.e. $\Sigma_{\mathbf{z}} \equiv E((\mathbf{z} - \bar{\mathbf{z}})(\mathbf{z} - \bar{\mathbf{z}})^T) = \mathbf{I}_n$ where $\bar{\mathbf{z}} = E(\mathbf{z})$ is the expected value of \mathbf{z} , and \mathbf{I}_n is the $n \times n$ identity matrix. In many standard ICA methods, the mean is also subtracted so that $\bar{\mathbf{z}} = \mathbf{0}$, but we must not subtract the mean for non-negative ICA, since this would lose any information about the positivity or negativity of sources [1]. If we decompose the covariance matrix of the observations $\Sigma_{\mathbf{x}} \equiv E((\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^T)$ into an eigenvector matrix \mathbf{E} and a diagonal eigenvalue matrix $\mathbf{D} = \text{diag}(d_1, \dots, d_n)$ to give $\Sigma_{\mathbf{x}} = \mathbf{E}\mathbf{D}\mathbf{E}^T$, for example using the Matlab function `eig`, then a typical choice for \mathbf{V} is

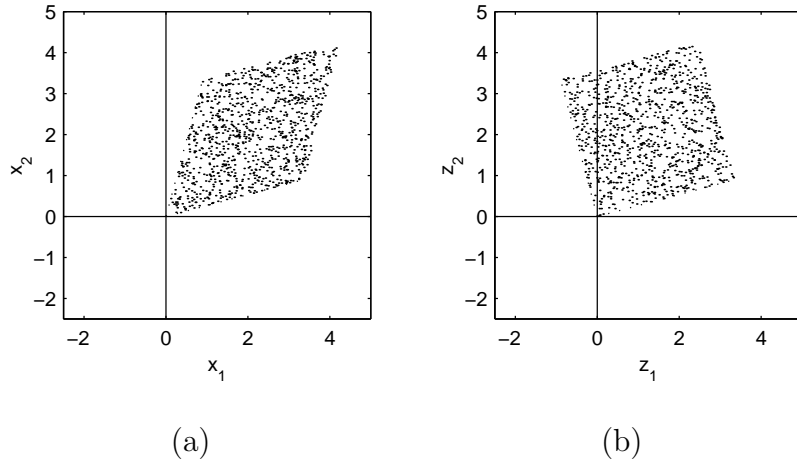


Fig. 1. Scatter plot for $n = 2$ inputs showing (a) original input data \mathbf{x} and (b) pre-whitened data \mathbf{z} .

$\mathbf{D}^{-1/2}\mathbf{E}^T$ [2]. For the remainder of this article we shall assume that our data has been pre-whitened like this, so that we have unit-variance pre-whitened vectors $\mathbf{z} = \mathbf{VAs}$.

Rotation The second stage is where the interesting geometry begins to appear. To visualize what we have achieved so far, see Fig. 1. We can clearly see from this diagram that the whitened data \mathbf{z} has solved part of our ICA task, by making the original source directions, shown as the edges of the scatter plot connected to the origin, orthogonal (at right angles) to each other. What remains is for us to *rotate* the whole system about the origin until all the data points are “locked” into the positive quadrant.

We would like to perform this rotation by finding a linear matrix \mathbf{W} which will give an output vector sequence $\mathbf{y} = \mathbf{Wz}$ which extracts the original sources \mathbf{s} . If we allowed all possible $n \times n$ matrices \mathbf{W} , each determined by n^2 real numbers, we would be searching over the space \mathbb{R}^{n^2} , the n^2 -dimensional real space. But since we assume that the sources have unit covariance $\Sigma_s \equiv E((\mathbf{s} -$

$\bar{\mathbf{s}})(\mathbf{s} - \bar{\mathbf{s}})^T) = \mathbf{I}_n$ we know that at a solution, the output \mathbf{y} must have unit covariance. Therefore we have

$$\mathbf{I}_n = \Sigma_{\mathbf{y}} \equiv E((\mathbf{y} - \bar{\mathbf{y}})(\mathbf{y} - \bar{\mathbf{y}})^T) = E(\mathbf{W}(\mathbf{z} - \bar{\mathbf{z}})(\mathbf{z} - \bar{\mathbf{z}})^T \mathbf{W}^T) = \mathbf{W}\mathbf{W}^T \quad (2)$$

and so the square matrix \mathbf{W} must be *orthogonal* at the solution, i.e. $\mathbf{W}^T = \mathbf{W}^{-1}$ so that $\mathbf{W}^T \mathbf{W} = \mathbf{W}\mathbf{W}^T = \mathbf{I}$. If \mathbf{W} is chosen such that $\Sigma_{\mathbf{y}} = \mathbf{I}$ and the elements y_i of \mathbf{y} are all non-negative, then our non-negative ICA problem is solved [1].

As is common with other ICA methods, we will not know the original variance of the sources, and we will not know which “order” they were in originally, so there will be a *scaling* and *permutation* ambiguity. However, in contrast to standard ICA, we will know the *sign* of the sources if we use a non-negative ICA approach, since the estimated sources must be non-negative. This means, for example, that solutions to image separation tasks will never produce “negative” images, as is possible for standard ICA models.

3 Cost function minimization using steepest descent search

To find the matrix \mathbf{W} that we need for our solution, it is often convenient to construct a cost function J so that we can express the task as an optimization problem. By constructing the derivative of J with respect to \mathbf{W} given by

$$\nabla_{\mathbf{W}} J \equiv \frac{\partial J}{\partial \mathbf{W}} \quad (3)$$

where $[\partial J / \partial \mathbf{W}]_{ij} = \partial J / \partial w_{ij}$, we can search for a direction in which a small change to \mathbf{W} will cause a decrease in J , and hence eventually reduce J to zero.

Probably the simplest adjustment to \mathbf{W} is to use the so-called *steepest-descent search* or *steepest gradient descent* method. Here we simple update \mathbf{W} at the k th step according to the scheme

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \eta \nabla_{\mathbf{W}} J \quad (4)$$

for some small update factor $\eta > 0$. This is sometimes written as $\mathbf{W}_{k+1} = \mathbf{W}_k + \Delta \mathbf{W}$ where $\Delta \mathbf{W} = -\eta \nabla_{\mathbf{W}} J$. For small η it approximates the ordinary differential equation (ode) $d\mathbf{W}/dt = -\eta \nabla_{\mathbf{W}} J$ which is known to reduce J since $dJ/dt = \langle (\nabla_{\mathbf{W}} J), (d\mathbf{W}/dt) \rangle = -\eta \|\nabla_{\mathbf{W}} J\|_F^2$ where the matrix inner product is given by $\langle \mathbf{A}, \mathbf{B} \rangle = \sum_{ij} a_{ij} b_{ij}$ and $\|\mathbf{M}\|_F = \sqrt{\sum_{ij} m_{ij}^2}$ is the (positive) Frobenius norm of \mathbf{M} .

For the non-negative ICA task we require the outputs y_i to be nonnegative at the solution. This therefore suggests a possible cost function [1]

$$J = \frac{1}{2} E(|\mathbf{y}_-|^2) = \frac{1}{2} E \left(\sum_i [\mathbf{y}_-]_i^2 \right) \quad \text{where} \quad [\mathbf{y}_-]_i = \begin{cases} y_i & \text{if } y_i < 0, \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

For easier comparison with other ICA algorithms, we sometimes write $\mathbf{y}_- = g(\mathbf{y})$ where $g(\mathbf{y}) = [g(y_1), \dots, g(y_n)]$ and $g(y_i) = y_i$ if $y_i < 0$ or 0 otherwise. Differentiating J in (5) while noting that $[\mathbf{y}_-]_i^2 = y_i [\mathbf{y}_-]_i$ yields $\partial J = E((\mathbf{y}_-)^T \partial \mathbf{y}) = E((\mathbf{y}_-)^T (\partial \mathbf{W}) \mathbf{z}) = \text{trace}(E(\mathbf{z}(\mathbf{y}_-)^T) \partial \mathbf{W})$ where $\text{trace}(\mathbf{A})$ is the the sum of the diagonal entries of \mathbf{A} , and where we used the identity $\text{trace}(\mathbf{AB}) = \text{trace}(\mathbf{BA})$ and the fact that $x = \text{trace}(x)$ for scalar x . From this we obtain

$$\nabla_{\mathbf{W}} J \equiv \frac{\partial J}{\partial \mathbf{W}} = E(\mathbf{y}_- \mathbf{z}^T) \equiv E(g(\mathbf{y}) \mathbf{z}^T) \quad (6)$$

for the derivative of J that we require.

3.1 Imposing the orthogonality constraint via penalty term

Although applying the steepest descent scheme (4) will tend to reduce the cost function J , we have so far ignored our orthogonality constraint on \mathbf{W} . A simple way to impose this would be to construct an additional cost or *penalty function* $J_{\mathbf{W}}$ which is zero whenever $\mathbf{W}\mathbf{W}^T = \mathbf{I}_n$, or positive otherwise. For example, we could use

$$J_{\mathbf{W}} = \frac{1}{2} \|\mathbf{W}\mathbf{W}^T - \mathbf{I}_n\|_F^2 = \frac{1}{2} \text{trace}((\mathbf{W}^T\mathbf{W} - \mathbf{I}_n)(\mathbf{W}^T\mathbf{W} - \mathbf{I}_n)) \quad (7)$$

which gives $J_{\mathbf{W}} \geq 0$ with $J_{\mathbf{W}} = 0$ if and only if $\mathbf{W}\mathbf{W}^T = \mathbf{I}_n$. Since we have $J \geq 0$ and $J_{\mathbf{W}} \geq 0$, the combined cost function

$$J_1 = J + \mu J_{\mathbf{W}} \quad (8)$$

for some $\mu > 0$ will also be positive, with $J_1 = 0$ when we have found the solution we require.

Differentiating $J_{\mathbf{W}}$ in (7) we get

$$\partial J_{\mathbf{W}} = \text{trace}((\mathbf{W}^T\mathbf{W} - \mathbf{I}_n)\partial(\mathbf{W}^T\mathbf{W})) \quad (9)$$

$$= 2\text{trace}((\mathbf{W}^T\mathbf{W} - \mathbf{I}_n)\mathbf{W}^T(\partial\mathbf{W})) \quad (10)$$

$$= 2\langle \mathbf{W}(\mathbf{W}^T\mathbf{W} - \mathbf{I}_n), \partial\mathbf{W} \rangle \quad (11)$$

giving $\nabla_{\mathbf{W}} J_{\mathbf{W}} \equiv \partial J_{\mathbf{W}} / \partial \mathbf{W} = 2\mathbf{W}(\mathbf{W}^T\mathbf{W} - \mathbf{I}_n)$. Therefore the modified update rule $\Delta\mathbf{W} = -\eta(\nabla_{\mathbf{W}} J + \mu \nabla_{\mathbf{W}} J_{\mathbf{W}}) = -\eta(\nabla_{\mathbf{W}} J + 2\mu\mathbf{W}(\mathbf{W}^T\mathbf{W} - \mathbf{I}_n))$ should be able to find a minimum of J such that \mathbf{W} is orthogonal.

For the non-negative ICA task we therefore have from (6)

$$\Delta\mathbf{W} = -\eta(E(\mathbf{y}_-\mathbf{z}^T) + 2\mu\mathbf{W}(\mathbf{W}^T\mathbf{W} - \mathbf{I}_n)) \quad (12)$$

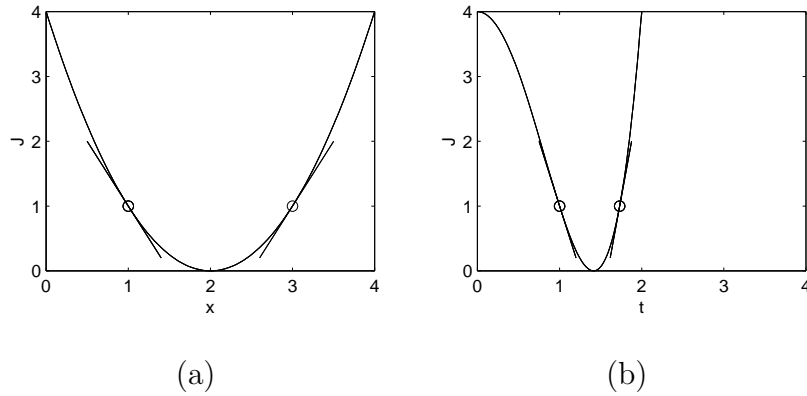


Fig. 2. Graph of (a) $J = (x - 2)^2$ and (b) $J = (t^2 - 2)^2$ showing change of apparent “steepness”.

for our steepest descent update with penalty term.

3.2 What is “steepest”?

But wait. What have we really done here? What does “steepest” mean in “steepest descent”?

Roughly speaking, *steepest* means *decreases the fastest per unit length*. In physical problems we know what length means, but are we sure that we have used the correct measure of length in eqn (4)? Do we even *know* what we have used as a measure of length?

To see why this might be a problem, instead of our n^2 -dimensional problem, let us consider the simpler curve

$$J = (x - 2)^2 \tag{13}$$

and suppose we wanted to decide which of the points $x = 1$ and $x = 3$ were steeper. Calculating the derivative with respect to x we get $dJ/dx = 2(x - 2)$ which gives $dJ/dx = -2$ at $x = 1$, and $dJ/dx = 2$ at $x = 3$. Therefore J has

equal “steepness” $|dJ/dx| = 2$ at both points, only in opposite directions.

But suppose now that x is actually a function of some other parameter t , so that $x = t^2$. So now $J = (t^2 - 2)^2$, and we can check that the value of J remains the same at corresponding values of $x = t^2$. However, for the derivative with respect to t we get $dJ/dt = 2(t^2 - 2)2t = 4t(t^2 - 2)$ so at $t = 1$ (corresponding to $x = 1$) we get $dJ/dt = -4 \cdot 1 = -4$, while for $t = \sqrt{3}$ (corresponding to $x = 3$) we get $dJ/dt = 4\sqrt{3} \cdot 1 = 4\sqrt{3}$. So J now seems steeper at $t = \sqrt{3}$ ($x = 3$) than at $t = 1$ ($x = 1$). What has happened?

The answer is that the way we measure *length* has changed. To see this, let us define an infinitesimal length $dl \equiv |dx|$ that measures how long dx is. We’ve chosen the most obvious definition of length here: simply the modulus of dx , so that it is always positive and $dl = 0$ if and only if $dx = 0$. So now we can calculate a measure of steepness properly:

$$\text{Steepness} = \frac{|\text{Change of } J|}{|\text{Length moved}|} = \frac{|dJ/dx|}{|dl/dx|} = |dJ/dx| = 2|x - 2| \quad (14)$$

since $dl/dx = +1$ if $dx > 0$ or -1 if $dx < 0$.

Now consider $t = x^2$, but keep the same measure of length $dl \equiv |dx| = |2tdt|$. We immediately notice that our length dl is now not just dependent on dt but increases as t increases. Recalculating our steepness using t we now get

$$\text{Steepness} = \frac{|dJ/dt|}{|dl/dt|} = \frac{|4t(t^2 - 2)|}{|2t|} = |2(t^2 - 2)|. \quad (15)$$

We can see that this is now the same, whether we calculated it using x or t . The lesson here is that we must be careful what we mean by *length* when we talk about “steepest descent”. Furthermore, it is not always obvious whether there is a *natural* measure for length or not, and it may depend on the application

you have in mind.

So, returning to our matrices, we now know that we are assuming a hidden length measure in $\nabla_{\mathbf{W}}J$ when we use that in steepest descent. It turns out that this is effectively $dl = \|d\mathbf{W}\|_F = \sqrt{\sum_{ij} dw_{ij}^2}$. As the usual Euclidean length of anything with n^2 components, this is probably the most obvious length to use, but can be sure it is the “best” one? What alternatives are there?

The key issue to spot here is what \mathbf{W} is *doing*. If it were being added to something, using a length $dl = \|d\mathbf{W}\|_F$ might be a reasonable thing to do. But in ICA, \mathbf{W} is being used to multiply something ($\mathbf{y} = \mathbf{W}\mathbf{z}$) so a particular change $d\mathbf{W}$ when \mathbf{W} is near zero (or near singular) will have larger relative effect than for large \mathbf{W} . Using geometric considerations related to this concept, Amari developed his *natural gradient* approach [3,4]. For the system considered here this gives the natural gradient

$$\nabla_{\text{nat}}J = (\nabla_{\mathbf{W}}J)\mathbf{W}^T\mathbf{W} \quad (16)$$

to be used in place of $\nabla_{\mathbf{W}}J$ in the update equation (4). (This is similar to the *relative gradient* approach of Cardoso and Laheld [5]).

For the non-negative ICA task substituting (6) in (16) we get $\nabla_{\text{nat}}J = E(\mathbf{y}_-\mathbf{z}^T)\mathbf{W}^T\mathbf{W} = E(\mathbf{y}_-\mathbf{y}^T)\mathbf{W}$ giving a natural gradient algorithm with penalty term of

$$\Delta\mathbf{W} = -\eta(E(\mathbf{y}_-\mathbf{y}^T) + 2\mu(\mathbf{W}\mathbf{W}^T - \mathbf{I}_n))\mathbf{W} \quad (17)$$

for the natural gradient version of our steepest descent update with penalty term. Notice that this is of the form $\Delta\mathbf{W} = \mathbf{H}\mathbf{W}$ for some square matrix \mathbf{H} : we will see a similar pattern also emerge later.

4 Constrained Optimization on Manifolds

In the previous section we considered the idea of performing steepest descent to minimize our cost function, and we have discovered that it is important to be clear by what we mean by *length*. So far we have assumed that we would allow \mathbf{W} free reign to roam over its entire n^2 -dimensional space \mathbb{R}^{n^2} .

However, we know that \mathbf{W} must be orthogonal at the solution, i.e. $\mathbf{W}^T \mathbf{W} = \mathbf{I}_n$, and it is also easy to start from an initial orthogonal matrix: $\mathbf{W}_0 = \mathbf{I}_n$ is orthogonal, for example. So can we arrange to constrain \mathbf{W} so that we *only* search over orthogonal matrices?

This leads to the idea of a *subset* of matrices that we want to search over, and is closely related to what Fiori calls *Orthonormal Strongly Constrained* (SOC) algorithms [6]. Instead of allowing any $\mathbf{W} \in \mathbb{R}^{n^2}$, we will only allow our matrix to move in this space of orthogonal matrices. The constraints mean that we can imagine this as some type of surface embedded inside our original space. Any matrix on the surface corresponds to an orthogonal matrix, while any matrix which is not on the surface is not orthogonal.

We can see that this surface containing the orthogonal matrices is a “smaller” space to search over, in that it has a lower dimensionality than the original one. If we write down the set of equations corresponding to the elements of $\mathbf{W}^T \mathbf{W} = \mathbf{I}_n$, we get $\sum_j w_{ji} w_{jk} = \delta_{ij}$ where the Kronecker delta is defined by $\delta_{ij} = 1$ if $i = j$, or 0 otherwise. We have $\sum_{i,k \leq n} 1 = n(n+1)/2$ different constraints, so the remaining space has dimension $n^2 - n(n+1)/2 = n(n-1)/2$.

But we have gained this smaller space at a cost: it now seems much more

difficult to work out how to “move about” in this space. In our original space \mathbb{R}^{n^2} of $n \times n$ matrices \mathbf{W} we could simply add any other $n \times n$ matrix $\Delta\mathbf{W} \in \mathbb{R}^{n^2}$ and we would still be in our space of matrices. But in our new, constrained space there are only certain movements we can make without breaking our constraint and moving away from the surface.

4.1 *Imposing constraints with a projection method*

One possible method to tackle this constraint problem would be to make the movements in two steps: First, update \mathbf{W} by any change $\Delta\mathbf{W} \in \mathbb{R}^{n^2}$; then modify \mathbf{W} to re-impose the constraint by *projecting* back onto our constraint surface. The projection could be performed using an orthogonalization method such as Gram-Schmidt orthogonalization [7,2]. This is a similar idea to the two-step version of Oja’s famous PCA neuron [8]. But this can lead to conditions where most of the update points away from the surface, and we spend almost all of our effort re-imposing our constraint, while not actually moving very far along the surface. We would therefore prefer if we could make sure that our updates before each projection step do not take us very far away from the constraint surface.

4.2 *Tangent updates*

Instead of allowing a general update direction to our matrix \mathbf{W} , we could restrict ourselves to update directions that attempt to maintain the orthogonality condition. For a continuous-time system with update $d\mathbf{W}/dt$, we would require $\mathbf{0} = d/dt(\mathbf{W}^T\mathbf{W} - \mathbf{I}_n) = d/dt(\mathbf{W}^T\mathbf{W}) = (d\mathbf{W}/dt)^T\mathbf{W} + \mathbf{W}^T(d\mathbf{W}/dt)$.

Now, since $\mathbf{W}\mathbf{W}^T = \mathbf{W}^T\mathbf{W} = \mathbf{I}_n$, we can write $d\mathbf{W}/dt = (d\mathbf{W}/dt)\mathbf{W}^T\mathbf{W} = \mathbf{H}\mathbf{W}$ where $\mathbf{H} = (d\mathbf{W}/dt)\mathbf{W}^T$. Then we have

$$0 = (d\mathbf{W}/dt)^T\mathbf{W} + \mathbf{W}^T(d\mathbf{W}/dt) = \mathbf{W}^T\mathbf{H}^T\mathbf{W} + \mathbf{W}\mathbf{H}\mathbf{W} = \mathbf{W}^T(\mathbf{H}^T + \mathbf{H})\mathbf{W}.$$

Pre- and post-multiplying by \mathbf{W} and \mathbf{W}^T respectively gives us $\mathbf{H}^T + \mathbf{H} = 0$, i.e. $\mathbf{H}^T = -\mathbf{H}$, meaning that \mathbf{H} must be *skew-symmetric*. This means we can also write $\mathbf{H} = \text{skew}((d\mathbf{W}/dt)\mathbf{W}^T) = \frac{1}{2}((d\mathbf{W}/dt)\mathbf{W}^T - \mathbf{W}(d\mathbf{W}/dt)^T)$ where $\text{skew}(\mathbf{M}) \equiv \frac{1}{2}(\mathbf{M} - \mathbf{M}^T)$ is the skew-symmetric (sometimes called “anti-symmetric”) part of \mathbf{M} . Any skew-symmetric matrix \mathbf{H} must have zeros on the diagonals, since the diagonal elements must satisfy $h_{ii} = -h_{ii} = 0$. Therefore \mathbf{H} has only $n(n-1)/2$ independent elements, confirming that $\mathbf{H}\mathbf{W}$ has enough degrees of freedom to “point” in the $n(n-1)/2$ different dimensions of our constrained surface.

If we had some unnormalized direction $d\mathbf{W}/dt$, we could find a version that is constrained to the surface as [9,10] $d\mathbf{W}/dt|_{\text{Orth}} = \mathbf{H}\mathbf{W}$ where we choose $\mathbf{H} = \frac{1}{2}((d\mathbf{W}/dt)\mathbf{W}^T - \mathbf{W}(d\mathbf{W}/dt)^T)$. We can also write

$$d\mathbf{W}/dt|_{\text{Orth}} = \frac{1}{2}((d\mathbf{W}/dt)\mathbf{W}^T\mathbf{W} - \mathbf{W}(d\mathbf{W}/dt)^T\mathbf{W}) \quad (18)$$

$$= \frac{1}{2}(d\mathbf{W}/dt - \mathbf{W}(d\mathbf{W}/dt)^T\mathbf{W}) \quad (19)$$

where the second line uses the fact that $\mathbf{W}^T\mathbf{W} = \mathbf{I}_n$. So for steepest descent search, where we normally set an update $\Delta\mathbf{W} = -\eta(\nabla_{\mathbf{W}}J)$ we could instead use

$$\Delta\mathbf{W} = -\frac{1}{2}\eta((\nabla_{\mathbf{W}}J)\mathbf{W}^T\mathbf{W} - \mathbf{W}(\nabla_{\mathbf{W}}J)^T\mathbf{W}) \quad (20)$$

or
$$\Delta\mathbf{W} = -\frac{1}{2}\eta((\nabla_{\mathbf{W}}J) - \mathbf{W}(\nabla_{\mathbf{W}}J)^T\mathbf{W}). \quad (21)$$

For the non-negative ICA task substituting (6) into (20), for example, we get

$$\Delta \mathbf{W} = -\eta(E(\mathbf{y}_- \mathbf{y}^T) - E(\mathbf{y} \mathbf{y}_-^T)) \mathbf{W} \quad (22)$$

which is the algorithm presented in [11].

4.3 Self-correcting tangent methods

So far so good. However, we now face another difficulty. It turns out that the surface containing the orthogonal matrices is *curved*, a bit like the surface of a globe. This means that even if we *did* start moving along a straight line which pointed “along” the surface at the point where we started (a line which is *tangent* to it), we would soon drift away from the surface as soon as we begin to move along the line [12].

This drift has been found to cause problems for many algorithms. Either we can occasionally re-impose the constraint via an orthogonalization method, or we can correct for this by adding an extra penalty term to correct the drift, similar to (12) and (17) we used earlier. Possible algorithms that result are then

$$\Delta \mathbf{W} = -\frac{1}{2} \eta ((\nabla_{\mathbf{W}} J) \mathbf{W}^T \mathbf{W} - \mathbf{W} (\nabla_{\mathbf{W}} J)^T \mathbf{W} + \mu \mathbf{W} (\mathbf{W}^T \mathbf{W} - \mathbf{I}_n)) \quad (23)$$

or

$$\Delta \mathbf{W} = -\frac{1}{2} \eta ((\nabla_{\mathbf{W}} J) - \mathbf{W} (\nabla_{\mathbf{W}} J)^T \mathbf{W} + \mu \mathbf{W} (\mathbf{W}^T \mathbf{W} - \mathbf{I}_n)). \quad (24)$$

Some earlier principal component analysis (PCA) algorithms were self-correcting like this “by accident”, and it took a while for researchers to realize why some of these were unstable when modified to extract minor components instead. See e.g. [13] for a discussion of deliberately self-correcting algorithms, including going beyond just the tangent method.

For the non-negative ICA task substituting (6) into (23), for example, we get

$$\Delta \mathbf{W} = -\frac{1}{2}\eta(E(\mathbf{y}_-\mathbf{y}^T) - E(\mathbf{y}\mathbf{y}_-^T)) + \mu(\mathbf{W}\mathbf{W}^T - \mathbf{I}_n)\mathbf{W}. \quad (25)$$

5 Imposing constraints using Lie groups

We have attempted to optimize our cost function J with our orthonormality constraint, but so far our weight matrix has stubbornly refused to “stay” on the constraint surface, drifting away eventually even if we use a tangent method. We have either had to “coax” it back with a penalty function, or “project” it back with an orthogonalization method. We would really like to be able to “lock” our matrices on to the constraint surface so that we never drift away from it. To do this we will need to understand a bit more clearly what it means by “move about” on this surface. To help us, we will digress a little to investigate what happens with constrained complex numbers, leading to the idea of a *Lie group*.

5.1 A simple Lie group: constrained complex numbers

Consider a pair of complex numbers, $z = x + iy$ and $w = u + iv$. If we wanted to add these, we would simply write $z + w = (x + u) + i(y + v)$, or to multiply we would get $zw = (ux - yv) + i(yu + xv)$. But multiplication is much easier if we represent our complex numbers in $r - \theta$ notation (Fig. 3). Specifically, if $z = re^{i\theta}$ and $w = se^{i\phi}$ for $r, s \geq 0$, then $zw = rse^{i(\theta+\phi)}$. This gives us a nice geometric visualization for what multiplication means: the modulus (length) is multiplied, $|zw| = rs$, and angles add, $\angle(zw) = \theta + \phi$.

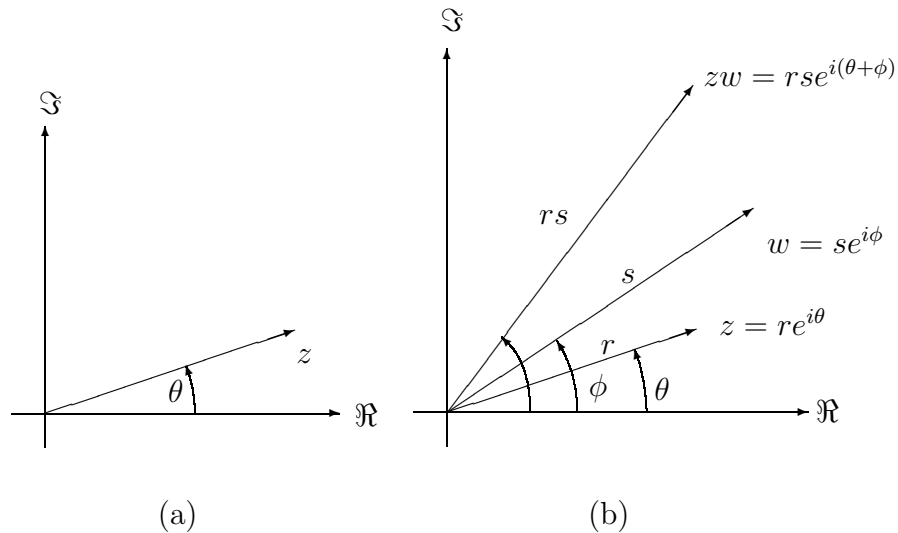


Fig. 3. Complex value (a) z in r - θ notation, and (b) product of z and w

Now, suppose we were only interested in the complex numbers with length 1, and we want to “move about” in this space of unit-length complex numbers. This is just the unit circle $r = 1$ in the Argand diagram (z -plane), a curved subset of the complex numbers. Any complex number z in this constrained set can be written $z = e^{i\theta}$ for some real θ . We know that z repeats every 2π , so we only need the range $0 \leq \theta < 2\pi$ (or any shifted version of this such as $-\pi \leq \theta < \pi$) to cover all the unit-length complex numbers of interest.

Now if we multiply $z = e^{i\theta}$ by $w = e^{i\phi}$ we get $zw = e^{i(\theta+\phi)}$, which is also a unit-length complex number. We can also see that if we wanted to get to any unit length complex number $y = e^{i\psi}$ by starting from $z = e^{i\theta}$ we can simply multiply z by $w = e^{i\phi}$ where we choose $\phi = \psi - \theta$.

We can recognize that the set of unit-length complex numbers, together with the operation “multiply”, forms a *group*. Let us remind ourselves of the requirements for a group G :

- (1) Closed under the operation: if $z, w \in G$, then $zw = y \in G$;
- (2) Associativity: $z(wy) = (zw)y$ for $z, w, y \in G$;

- (3) Identity element: $I \in G$, such that $Iz = zI = z$;
- (4) Each element has an inverse: z^{-1} such that $z^{-1}z = zz^{-1} = I$;

We can easily check these apply to unit-length complex numbers, and in particular we get the identity $I = 1 = e^{i0}$ and the inverse of $z = e^{i\theta}$ is given by $z^{-1} = e^{-i\theta}$. We can call this a commutative, or *Abelian*, group, since multiplication of complex numbers commutes: $zw = wz$. (Recall that multiplication is not commutative for matrices, i.e. $\mathbf{AB} \neq \mathbf{BA}$ for two matrices \mathbf{A} and \mathbf{B} in general).

At this point we realize that the addition operator between unit-length complex numbers is of no use to us anymore, since it will almost always take us away from our constraint surface, the unit circle. However, we do notice that *multiplication of complex numbers* z has an interesting correspondence with *addition of angles* θ : we will see this correspondence crop up again when we return to consider our constrained matrices.

Our set of unit-length complex numbers, with the multiplication operator, also has the property that it is “smooth”. If we look at a very local region of our group it “looks like” a region of the real line \mathbb{R} , albeit slightly curved. This means that we can use real-valued coordinates (for example, the angle θ), to describe a path over a local region of our group. Without going into the technical details (see e.g. [14] or a suitable text book for these) this “smoothness” means that it is a *Lie group*, named after Sophus Lie (pronounced “Lee”), a famous 19th century Norwegian mathematician. Lie groups are particularly important in mathematics and physics, such as the theory of general relativity, or more recently in robotics and computer vision.

The ability to put a local system of coordinates in \mathbb{R}^n for some n , onto any

local region of our group, also means that these unit-length complex numbers form what is called a *manifold*. We talk about a “local” coordinate system, because it is quite unusual to be able to wrap a single coordinate system on to our points without something going funny somewhere. Imagine you have a very long piece of string, representing the real line \mathbb{R} , and you want to wrap it on to the unit-length complex numbers on the z -plane. You will find that you have to overlap the string in some region, otherwise the coordinate system would have a break in it. Where the string overlaps we have a choice of coordinates, but that isn’t a problem: we just end up with equivalent coordinates. For example, if the angle θ is our coordinate, the coordinates $\theta = \pi/6$ and $\theta = \pi/6 + 2\pi$ give the same point $z = e^{i\pi/6} = e^{i(\pi/6+2\pi)}$. The fact that we can use a one-dimensional piece of string means that we have a one-dimensional manifold.

More complex manifolds may need more than one coordinate system. For two-dimensional manifolds, like a sphere or a torus (doughnut), we can visualize the process of fitting these coordinate systems to be like cutting out pieces of rubber sheeting, with the horizontal and vertical lines of our coordinate system marked on, and sticking these on to the manifold. If we make sure we never leave any gaps which are not covered by any rubber sheets, and we always smooth out any folds, we will have a smooth (*differentiable*) manifold. Mariners and cartographers have known this about the terrestrial globe for centuries, in the way that they have pictured the curved globe on flat paper. For navigating in detail, they use a series of flat *charts*, each of which is a *map* placing a 2-dimensional coordinate system (the position on the paper) onto a part of the surface of the globe. If the collection of charts is complete, every small region on the globe will be mapped onto a small region on at least one of these charts, with some overlap between the charts, and no parts of the

globe missing. A complete collection of charts could be bound together into an *atlas*. The mathematical names for these concepts are no coincidence.

5.2 Differentiating on a manifold

The smoothness of our group of unit-length complex numbers means that we can think about making very small changes. We are used to this when we differentiate functions of real numbers: for example, the derivative of a function $f(x)$ at x is defined to be the limit

$$\frac{df}{dx} = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{(x + \Delta x) - x} = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \quad (26)$$

where we can see that the difference between $x + \Delta x$ and x is due to the addition of Δx , the movement from x from $x + \Delta x$. On our manifold, we don't have addition of *elements* anymore, but we can change the *coordinates* by adding something small to them.

For example, to make a small change to a unit-length complex number $z = e^{i\theta}$ we can add a small angle $\Delta\theta$ to get $z' = e^{i(\theta+\Delta\theta)} = e^{i\Delta\theta}e^{i\theta}$. We can also see that this is equivalent to multiplying by a unit-length complex number $w = e^{i\Delta\theta}$ which is very close to 1 (the identity). By the definition of $e^x = 1 + x + x^2/2! + \dots$ we get $w = 1 + i\Delta\theta + (\Delta\theta)^2/2 + \dots \approx 1 + i\Delta\theta$ giving $z' \approx (1 + i\Delta\theta)z = z + i(\Delta\theta)z$ This allows us to find the derivative of $z(\theta)$ with respect to θ :

$$\frac{d}{d\theta}z = \lim_{\Delta\theta \rightarrow 0} \frac{z(\theta + \Delta\theta) - z(\theta)}{\Delta\theta} = \frac{z + i(\Delta\theta)z - z}{\Delta\theta} = iz. \quad (27)$$

Thus the derivative of any z in our group is the same as multiplication by i , or rotation by $\pi/2$ in the z -plane.

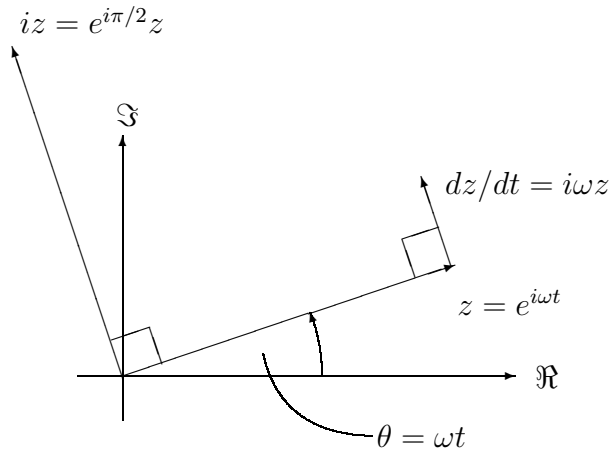


Fig. 4. Derivative of $z = \exp(i\omega t)$

While in this case the derivative happened to have unit length, this is not true in general. For example, suppose the angle evolves with some angular velocity ω , giving $\theta = \omega t$. Then we have $z = e^{i\omega t}$, giving a rate of change for $dz/dt = i\omega z = \omega e^{i(\omega t + \pi/2)}$ which is not in the group: derivatives are not constrained to the unit length group. However, the derivatives are constrained to point “along” the surface at the point where we are differentiating, so at the point z the derivatives are always *tangent to the manifold at z* . We can say that any vector $dz/dt = i\omega z$ for a particular ω is a *tangent vector* at z , and we call the space of all possible tangent vectors at z the *tangent space T_z* at z . We can see that the direction that our tangent vectors point along depends on where we are on our manifold, so we always have to say it is a tangent “at z ”.

If we were working on the usual Euclidean straight line of points $x \in \mathbb{R}$, our tangents always point along the straight line wherever we are. This means the tangent spaces are the same everywhere, so in Euclidean space we don’t have to bother to say “at x ” all the time. This is another reason (as well as the “length” issue) that we have to be more careful about derivatives in curved manifolds than we are used to in Euclidean space.

Well this is all very interesting, but what has this got to do with matrices? We will see that the set of orthogonal 2×2 matrices also forms a group and manifold which is very similar to the unit-length complex numbers, so a lot of these ideas will carry over to these. And more generally, we will see later that orthogonal $n \times n$ matrices can be transformed into a *block diagonal* form, composed of blocks of orthogonal 2×2 matrices. Knowing how these 2×2 work is key to our problem of optimization with an orthogonality constraint.

6 The Lie group of orthogonal matrices

Let us return to our orthogonal matrices with real entries. These also turn out to be a group, with matrix multiplication as the operator: we give these groups the label $O(n)$, for Orthogonal, $n \times n$ matrices.

We can check that $O(n)$ really is a group for any $n \geq 1$. For example, if \mathbf{W} and \mathbf{Z} are orthogonal, so that $\mathbf{W}^T \mathbf{W} = \mathbf{I}_n$ and similarly for \mathbf{Z} , then for $\mathbf{V} = \mathbf{WZ}$ we get $\mathbf{V}^T \mathbf{V} = \mathbf{Z}^T \mathbf{W}^T \mathbf{WZ} = \mathbf{Z}^T \mathbf{Z} = \mathbf{I}_n$ so \mathbf{V} is also orthogonal. We have an identity element \mathbf{I}_n making $\mathbf{I}_n \mathbf{W} = \mathbf{W} \mathbf{I}_n = \mathbf{W}$, each element \mathbf{W} has an inverse $\mathbf{W}^{-1} = \mathbf{W}^T$, and matrix multiplication is associative, as required for the group operation.

Let us quickly consider $O(1)$, the 1×1 “matrices” (really scalars) w which are orthogonal, i.e. $1 = w^T w = w^2$, so either $w = 1$ or $w = -1$. This is the 2-element group $\{1, -1\}$ with the multiplication operator. From the preceding argument we know this is a group. We can also check this directly, since e.g. $1 \times -1 = -1 \times 1 = -1$, the identity element is 1, we have inverses such as $(-1)^{-1} = -1$, and so on. However, we can’t smoothly get from one element

to the other, so this is a *discrete* group.

So let us try $n = 2$, the group $O(2)$ of 2×2 orthogonal matrices. In independent component analysis, this will correspond to the simplest useful case of $n = 2$ observations (e.g. 2 microphones) from which we want to estimate 2 sources. Imposing the $n(n+1)/2 = 3$ conditions implied by the equation $\mathbf{W}^T \mathbf{W} = \mathbf{I}_n$, we eventually find we can express \mathbf{W} as one of the following two types:

$$\text{type (a) } \mathbf{W} = \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix} \quad \text{or type (b) } \mathbf{W} = \begin{pmatrix} \cos \theta & -\sin \theta \\ -\sin \theta & -\cos \theta \end{pmatrix} \quad (28)$$

for some real θ . From any \mathbf{W} of type (a) we can smoothly get to any other \mathbf{W} of type (a) simply by varying θ , and similarly from a \mathbf{W} of type (b) we can get to any other of type (b) by varying θ . So it looks like a Lie group and a manifold, in that it is smooth and we can move around by varying θ . But the existence of the two types of \mathbf{W} is still a bit puzzling.

To see what is happening, take a look at the determinant. For orthogonal matrices we must have $(\det \mathbf{W})^2 = \det(\mathbf{W}^T \mathbf{W}) = \det \mathbf{I}_n = 1$, so $\det \mathbf{W} = \pm 1$. For type (a) matrices, we have $\det \mathbf{W} = \cos^2 \theta + \sin^2 \theta = 1$, while for type (b) we have $\det \mathbf{W} = -\cos^2 \theta - \sin^2 \theta = -1$. We can always get from type (a) to type (b) and vice versa by multiplying by a type (b) matrix (with determinant -1), but we can never *smoothly* vary the determinant between 1 and -1 .

Therefore the group $O(2)$ actually consists of two disconnected pieces, or *components*: we therefore say it is *disconnected*. We can smoothly move about in either of these, but to move from one to the other requires a “jump” from one component to the other. To visualize what is going on here, imag-

ine a matrix of type (a) is specified by two unit-length orthogonal vectors, $\mathbf{w}_1 = (\cos \theta, -\sin \theta)^T$, $\mathbf{w}_2 = (\sin \theta, \cos \theta)$ lying in a two-dimensional space (x_1, x_2) . We could say these form a “right-handed set”, since we require a positive turn of $+\pi/2$ to go from \mathbf{w}_1 to \mathbf{w}_2 . However, a matrix of type (b), represented by $\mathbf{w}_1 = (\cos \theta, -\sin \theta)^T$, $\mathbf{w}_2 = (\sin \theta, \cos \theta)$ forms a “left-handed set”, implying a turn of $-\pi/2$ from \mathbf{w}_1 to \mathbf{w}_2 . To get from one to the other, we would have to “flip over” the pair of vectors and place them down again “the other way round”.

We really do not want this extra complication. We would much rather have our matrices forming a *connected* Lie group, one with just a single component where we can get smoothly between any two points. We will therefore consider just the subgroup $\text{SO}(n)$ of orthogonal matrices with determinant 1, which is a connected subgroup: we just omit all the matrices of type (b), which had determinant -1 .

6.1 The special orthogonal group $\text{SO}(n)$

So now we are left with just the matrices of type (a):

$$\mathbf{W} = \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix}. \quad (29)$$

We know that these form a group, since multiplication preserves orthogonality and determinant 1. But what happens to the angles θ ? By some tedious

trigonometry (or otherwise), we can confirm that

$$\begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{pmatrix} = \begin{pmatrix} \cos(\theta + \phi) & \sin(\theta + \phi) \\ -\sin(\theta + \phi) & \cos(\theta + \phi) \end{pmatrix} \quad (30)$$

so that multiplication of matrices corresponds to addition of the angles θ . This also tells us that multiplication of matrices in $\text{SO}(2)$ is commutative, so $\text{SO}(2)$ is an Abelian (commutative) group. We will see later that this is a bit of a special case: the groups $\text{SO}(n)$ with $n > 2$ are not Abelian.

By now we have probably noticed a correspondence between the group $\text{SO}(2)$ of special (determinant 1) orthogonal 2×2 matrices and the group of unit-length complex numbers we considered earlier. They both operate in exactly the same way, with matrix multiplication over $\text{SO}(2)$ corresponding to complex multiplication in the unit-length complex numbers, and can both be specified completely by an angle $0 \leq \theta < 2\pi$. Two groups that “act the same” like this are said to be *isomorphic*.

Both of these are also isomorphic to the group of angles $\theta \in [0, 2\pi) = \{\theta | 0 \leq \theta < 2\pi\}$, with the operator of addition modulo 2π . So we now have an easy way to move about in the group $\text{SO}(2)$ of 2×2 special (determinant 1) orthogonal matrices. For each movement, we can proceed as follows:

- (1) Given some $\mathbf{W} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \in \text{SO}(2)$, calculate $\theta = \arctan(s, c)$
- (2) Move from angle θ to a new angle $\theta' = \theta + \Delta\theta$
- (3) Transform to the new matrix $\mathbf{W}' = \begin{pmatrix} \cos \theta' & \sin \theta' \\ -\sin \theta' & \cos \theta' \end{pmatrix} \in \text{SO}(2)$.

In step 1, $\arctan(\cdot, \cdot)$ is a four-quadrant arc tangent function defined such that $\arctan(\sin \theta, \cos \theta) = \theta$.

As an alternative, we can avoid having to calculate the arc tangent by realizing that addition in the θ domain is equivalent to multiplication in $SO(2)$. We can therefore make each step as follows:

- (1) From the desired movement $\Delta\theta$, calculate the (multiplicative) update matrix $\mathbf{R} = \begin{pmatrix} \cos(\Delta\theta) & \sin(\Delta\theta) \\ -\sin(\Delta\theta) & \cos(\Delta\theta) \end{pmatrix}$.
- (2) Update \mathbf{W} to the new matrix $\mathbf{W}' = \mathbf{R}\mathbf{W} \in SO(2)$

We will find that this second method turns out to be quite convenient for $n \geq 3$, particularly since the equivalent of “addition of angles” no longer works quite as easily for $SO(n)$ with $n \geq 3$.

6.2 Derivatives of \mathbf{W} and the matrix exponential

Let us now consider what happens when we make very small changes to \mathbf{W} . We were able to find a derivative on our manifold of unit-length complex numbers, so let us try this with these orthonormal matrices. If we let $\theta = t\phi$ and differentiate $\mathbf{W} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$ with respect to t , we get

$$\frac{d}{dt}\mathbf{W} = \begin{pmatrix} -\sin\theta & \cos\theta \\ -\cos\theta & -\sin\theta \end{pmatrix} \cdot \phi = \begin{pmatrix} 0 & \phi \\ -\phi & 0 \end{pmatrix} \mathbf{W}. \quad (31)$$

We can also check that the constraint is not violated as t changes, i.e. that $\frac{d}{dt}(\mathbf{W}^T\mathbf{W} - \mathbf{I}_n) = 0$. Expanding this we get $\mathbf{W}^T(d\mathbf{W}/dt) + (d\mathbf{W}/dt)^T\mathbf{W} = \mathbf{W}^T \begin{pmatrix} 0 & \phi \\ -\phi & 0 \end{pmatrix} \mathbf{W}^T + \mathbf{W}^T \begin{pmatrix} 0 & -\phi \\ \phi & 0 \end{pmatrix} \mathbf{W}^T = \mathbf{0}$ as required.

Now we know that for scalars, if $dz/dt = bz$ then $z = e^{tb}$. We can confirm that

this also works for matrices. The matrix exponent of $t\mathbf{B}$ is, by definition

$$\exp(t\mathbf{B}) = \mathbf{I} + t\mathbf{B} + \frac{t^2\mathbf{B}^2}{2!} + \cdots + \frac{t^k\mathbf{B}^k}{k!} + \cdots \quad (32)$$

from which is is straightforward to verify that

$$\frac{d}{dt} \exp(t\mathbf{B}) = \mathbf{0} + \mathbf{B} + \mathbf{B} \frac{t\mathbf{B}}{1!} + \cdots + \mathbf{B} \frac{t^{k-1}\mathbf{B}^{k-1}}{(k-1)!} + \cdots = \mathbf{B} \exp(t\mathbf{B}). \quad (33)$$

We must be rather careful with this matrix exponential. For example, since matrix multiplication is not commutative, $\exp(\mathbf{A} + \mathbf{B}) \neq \exp(\mathbf{A})\exp(\mathbf{B})$ in general. Therefore \mathbf{W} in (31) is of the form $\mathbf{W} = \exp(t\Phi)$ where $\Phi = \begin{pmatrix} 0 & \phi \\ -\phi & 0 \end{pmatrix}$, or alternatively (since $\theta = t\phi$) we get

$$\mathbf{W} = \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix} = \exp \Theta \quad \text{where} \quad \Theta = \begin{pmatrix} 0 & \theta \\ -\theta & 0 \end{pmatrix} \quad (34)$$

Therefore the tangent space $T_{\mathbf{W}}$ to the manifold $\text{SO}(2)$ at \mathbf{W} is formed by matrices of the form $\Psi\mathbf{W}$ where $\Psi^T = -\Psi$ is skew-symmetric.

One of the nice properties of $\text{SO}(2)$ is that, given any skew-symmetric $\Phi \neq \mathbf{0}$, all of the elements in the group can be specified by a single real parameter t , specifically as $\mathbf{W}(t) = \exp(t\Phi)$. We can therefore call $\text{SO}(2)$ a “one-parameter group”.

6.3 Optimization over $\text{SO}(2)$

Let us return now to our original problem of nonnegative ICA: we have some cost function $J(\mathbf{W})$ that we want to minimize over the manifold of orthogonal matrices. Since we want to change \mathbf{W} slowly, we will only consider the group $\text{SO}(n)$ of special (determinant 1) orthogonal matrices. We should make sure

we will not miss out any important solutions in this way: does it matter that we only allow “left-handed” solutions rather than “right-handed” solutions?

For the nonnegative ICA task, we know that any solutions we find will be some permutation of the sources. If we restrict ourselves to searching over $SO(n)$ rather than both components of the disconnected group $O(n)$, this simply means that we will only be able to find half of the permutations of the sources: either all the even permutations, or all of the odd permutations (depending on whether the processing that took place before we apply our matrix had positive or negative determinant). However this is not a problem for us, since any permutation is as good as any other. (For the standard ICA problem, a solution with a negative sign flip will also change the determinant, so only half of the combinations of sign flips and permutations will be allowed. Again this is not a problem, since any permutation and combination of sign flips is an acceptable solution for standard ICA).

So we can now design an algorithm to find a minimum of $J(\mathbf{W})$. Since all matrices $\mathbf{W} \in SO(2)$ can be parameterized by a single parameter u in $\mathbf{W} = \exp(u\Phi)$ for some constant skew-symmetric matrix Φ , we simply need to calculate dJ/du and change u to reduce J . If we were working in continuous time t , we might use an ordinary differential equation (ode), setting $du/dt = -\eta dJ/du$ which would then guarantee that $dJ/dt = dJ/du \cdot du/dt = -\eta(dJ/du)^2$ was negative, or zero when $dJ/du = 0$.

The discrete time version of this is the “steepest descent” algorithm $u(t+1) = u(t) + \Delta u$ where $\Delta u = -\eta dJ/du$. This update for Δu is equivalent to a multiplicative change to \mathbf{W} of $\mathbf{W}(t+1) = \mathbf{R}\mathbf{W}(t)$ where $\mathbf{R} = \exp(\Delta u \cdot \Phi)$.

6.4 Optimization by Fourier expansion

Using an ode is not the only way to find a minimum of a function $J(u)$ along the line $\{u\}$: we would use any one of many line search methods [15]. But we also know that $J(u)$ repeats every time $u\Phi = \begin{pmatrix} 0 & 2\pi \\ -2\pi & 0 \end{pmatrix}$, or if we set $\Phi = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$, $J(u)$ repeats every time $u = 2k\pi$ for integer k .

Suppose $J(u)$ were not cyclical. We could approximate it using a Taylor expansion around its minimum u^* , giving

$$J(u) = \sum_{k=0}^{\infty} a_k (u - u^*)^k \approx a_0 + a_1(u - u^*) + a_2(u - u^*)^2. \quad (35)$$

Calculating derivatives of our approximation we have $J'(u) \approx a_1 + 2a_2(u - u^*)$ and $J''(u) \approx 2a_2$. Now, we know that $J'(u^*) = 0$ since it is the minimum, so $a_1 = 0$, and therefore we can conclude that $J'(u) \approx 2a_2(u - u^*)$ and hence $J'(u)/J''(u) \approx (u - u^*)$ leading to $u^* \approx u - J'(u)/J''(u)$ which is the familiar Newton update step.

Since $J(u)$ is cyclical, we can instead use a Fourier expansion around $u = u^*$, giving:

$$J(u) = a_0 + \sum_{k=1}^{\infty} a_k \cos(k(u - u^*)) + \sum_{k=1}^{\infty} b_k \sin(k(u - u^*)) \quad (36)$$

$$\approx a_0 + a_1 \cos(u - u^*) + b_1 \sin(u - u^*). \quad (37)$$

Taking derivatives as above gives us $J'(u) \approx -a_1 \sin(u - u^*) + b_1 \cos(u - u^*)$ but again we realize that $J'(u) = 0$ at the minimum $u = u^*$ so $b_1 \approx 0$, and we can simplify this to $J'(u) \approx -a_1 \sin(u - u^*)$. Differentiating again we get $J''(u) \approx -a_1 \cos(u - u^*)$ so $J'(u)/J''(u) \approx \tan(u - u^*)$ and hence $u - u^* \approx \arctan(J'(u), J''(u))$ where $\arctan(s, c)$ is the 4-quadrant arc tangent

function such that $\arctan(\sin \theta, \cos \theta) = \theta$ for $0 \leq \theta < 2\pi$. This therefore suggests that we should step to $u^* \approx u - \arctan(dJ/du, d^2J/du^2)$. For small updates, this approximates the usual Newton step [16].

7 Optimization over $\text{SO}(n)$ for $n > 2$: The commutation problem

We know that in general multiplication of real and complex scalars commutes ($wz = zw$) but for matrices this is not true in general, i.e. $\mathbf{AB} \neq \mathbf{BA}$. We have seen that matrices in $\text{SO}(2)$ do commute: for $\mathbf{W}, \mathbf{Z} \in \text{SO}(2)$ we have $\mathbf{W} = \exp(t_1\Psi)$ and $\mathbf{Z} = \exp(t_2\Psi)$ for some fixed skew-symmetric Ψ , then $\mathbf{WZ} = \exp((t_1 + t_2)\Psi) = \mathbf{ZW}$.

In fact, we know the following: multiplication of two matrices \mathbf{A}, \mathbf{B} is commutative if and only if they share all eigenvectors [7]. This “if” direction is easy to see if the eigenvector decomposition exists: since $\mathbf{AE} = \mathbf{E}\Lambda_{\mathbf{A}}$ we can decompose $\mathbf{A} = \mathbf{E}\Lambda_{\mathbf{A}}\mathbf{E}^{-1}$ and $\mathbf{B} = \mathbf{E}\Lambda_{\mathbf{B}}\mathbf{E}^{-1}$ where Λ_{α} are diagonal eigenvalue matrices and \mathbf{E} is the (shared) matrix of eigenvectors, giving $\mathbf{AB} = \mathbf{E}\Lambda_{\mathbf{A}}\Lambda_{\mathbf{B}}\mathbf{E}^{-1} = \mathbf{E}\Lambda_{\mathbf{B}}\Lambda_{\mathbf{A}}\mathbf{E}^{-1} = \mathbf{BA}$. Note that if either \mathbf{A} or \mathbf{B} both have repeated eigenvalues, we are free to choose \mathbf{E} from the set of possible eigenvectors if necessary. Thus if we deliberately constructed \mathbf{A} and \mathbf{B} to share their eigenvectors, we would have made sure that they commuted.

7.1 Non-commutation of $\text{SO}(3)$

The first time we hit the commutation problem is for $\text{SO}(3)$, the group of 3×3 special (determinant 1) orthogonal matrices. Much as matrices in $\text{SO}(2)$ correspond to rotations about the origin in the 2-dimensional real space \mathbb{R}^2 , el-

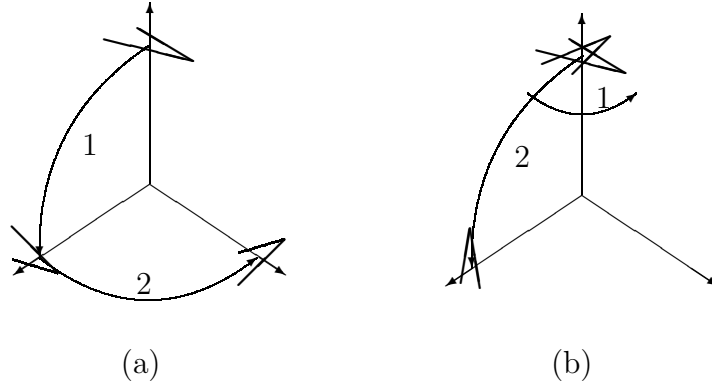


Fig. 5. Multiplication by matrices in $SO(3)$, corresponding to rotations about an axis in 3-space, are not commutative. Starting from a wedge at the top of the diagram pointing to the right, (a) illustrates rotation first about a horizontal axis followed by rotation about a vertical axis, while (b) illustrates rotation first about the vertical axis followed by the horizontal axis.

elements of $SO(3)$ correspond to rotations about the origin in the 3-dimensional real-space \mathbb{R}^3 . To illustrate, see Figure 5. We can see that rotations about different axes do not commute. For an illustration in matrix notation, using elements from $SO(3)$, we have

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix} \quad \text{but} \quad (38)$$

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{pmatrix}. \quad (39)$$

7.2 Lie brackets and Lie algebras

This non-commutativity is also true for small rotations. For skew-symmetric square matrices \mathbf{A} and \mathbf{B} , let us see what happens if we multiply the near-identity matrices $\exp(\epsilon\mathbf{A})$ and $\exp(\epsilon\mathbf{B})$ for small scalar ϵ . Specifically, we want to find the difference between $\exp(\epsilon\mathbf{A})\exp(\epsilon\mathbf{B})$ and $\exp(\epsilon\mathbf{B})\exp(\epsilon\mathbf{A})$. For this difference, we get

$$\begin{aligned} & \exp(\epsilon\mathbf{A})\exp(\epsilon\mathbf{B}) - \exp(\epsilon\mathbf{B})\exp(\epsilon\mathbf{A}) \\ &= \left(\mathbf{I} + \epsilon\mathbf{A} + \frac{1}{2}\epsilon^2\mathbf{A}^2\right) \left(\mathbf{I} + \epsilon\mathbf{B} + \frac{1}{2}\epsilon^2\mathbf{B}^2\right) \\ & \quad - \left(\mathbf{I} + \epsilon\mathbf{B} + \frac{1}{2}\epsilon^2\mathbf{B}^2\right) \left(\mathbf{I} + \epsilon\mathbf{A} + \frac{1}{2}\epsilon^2\mathbf{A}^2\right) + O(\epsilon^3) \end{aligned} \quad (40)$$

$$= \epsilon^2[\mathbf{A}, \mathbf{B}] + O(\epsilon^3) \quad (41)$$

where the matrix commutator $[\cdot, \cdot]$ is defined by $[\mathbf{A}, \mathbf{B}] = \mathbf{AB} - \mathbf{BA}$. Now, we see that $[\mathbf{A}, \mathbf{B}]^T = (\mathbf{AB} - \mathbf{BA})^T = \mathbf{B}^T\mathbf{A}^T - \mathbf{A}^T\mathbf{B}^T = \mathbf{BA} - \mathbf{AB} = -[\mathbf{A}, \mathbf{B}]$ since \mathbf{A} and \mathbf{B} are skew-symmetric. Therefore the result of applying this matrix commutator to skew-symmetric matrices is itself skew-symmetric, so the set of skew-symmetric matrices is *closed* under the matrix commutator.

This set of skew-symmetric matrices with the matrix commutator is an example of what is called a *Lie algebra*. Firstly it is a *vector space*, meaning that we can add skew symmetric matrices \mathbf{A} and \mathbf{B} to get another skew-symmetric matrix $\mathbf{C} = \mathbf{A} + \mathbf{B}$, and we can multiply by scalars, so $a\mathbf{B}$ is also skew-symmetric. (The fact that each of the objects \mathbf{A} and \mathbf{B} is actually a “matrix” does not stop this from being a “vector” space for this purpose. If you prefer, simply imagine forming a vector by concatenating the $n(n-1)/2$ independent entries in the upper half of the matrix to form a column vector with $n(n-1)/2$ entries). Secondly, it has a binary operation $[\cdot, \cdot]$ called the *Lie bracket*, which has the following properties:

$$[\mathbf{A}, \mathbf{A}] = \mathbf{0} \quad (42)$$

$$[\mathbf{A} + \mathbf{B}, \mathbf{C}] = [\mathbf{A}, \mathbf{C}] + [\mathbf{B}, \mathbf{C}] \quad (43)$$

$$[\mathbf{A}, [\mathbf{B}, \mathbf{C}]] + [\mathbf{B}, [\mathbf{C}, \mathbf{A}]] + [\mathbf{C}, [\mathbf{A}, \mathbf{B}]] = \mathbf{0} \quad (44)$$

The first two of these properties imply that $[\mathbf{A}, \mathbf{B}] = -[\mathbf{B}, \mathbf{A}]$. The third property is called the *Jacobi identity* and embodies the notion that the Lie bracket is concerned with a derivative-type action: notice the similarity with the scalar expression $\frac{d}{dx}(yz) = y(\frac{d}{dx}z) + z(\frac{d}{dx}y)$.

Actually, we can make any vector space into a (somewhat trivial) Lie algebra, just by defining $[z, w] = 0$ for any pair of elements z, w in the vector space. This would be an *Abelian* (commutative) Lie algebra, since the zero bracket implies $zw = wz$. Actually we have already met one of these Abelian Lie algebras: the 2×2 skew-symmetric matrices with this matrix commutator as the Lie bracket form a Lie algebra called $\mathfrak{so}(2)$ (gothic letters are often used to denote Lie algebras). We have seen that all matrix pairs $\mathbf{A}, \mathbf{B} \in \mathfrak{so}(2)$ commute, so $[\mathbf{A}, \mathbf{B}] = \mathbf{0}$ and hence this is an Abelian (commutative) Lie algebra. Unsurprisingly, the skew-symmetric matrices in the Lie algebra $\mathfrak{so}(2)$ are related to those in the Lie group $SO(2)$: the *exponential* of a matrix $\mathbf{B} \in \mathfrak{so}(2)$ is a matrix in $SO(2)$, i.e. $\mathbf{B} \in \mathfrak{so}(2) \mapsto \exp(\mathbf{B}) \in SO(2)$.

Being able to work in the space of Lie algebras gives the key to so-called *Lie group methods* to solve differential equations that evolve on a Lie group manifold [17]. The basic method works like this (Fig. 6):

- (1) Use the logarithm (inverse of $\exp(\cdot)$) to map from an element in the Lie group $\mathbf{W} \in SO(n)$ to one in the Lie algebra $\mathbf{B} \in \mathfrak{so}(n)$.
- (2) Move about in $\mathfrak{so}(n)$ to a new $\mathbf{B}' \in \mathfrak{so}(n)$
- (3) Use $\exp(\cdot)$ to map back into $SO(n)$, giving $\mathbf{W} = \exp(\mathbf{B}')$.

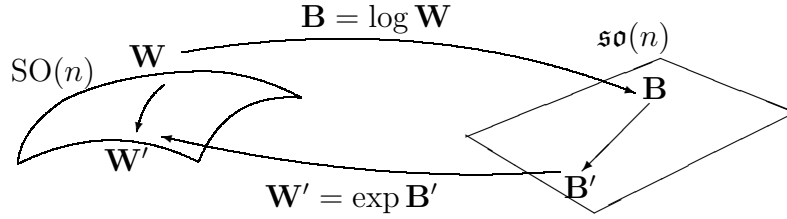


Fig. 6. Motion from \mathbf{W} on the Lie group $SO(n)$ by mapping \mathbf{B} to the Lie algebra $\mathfrak{so}(n)$, moving to $\mathbf{B}' \in \mathfrak{so}(n)$, and mapping back to $\mathbf{W}' \in SO(n)$.

Since $\mathfrak{so}(n)$ is a vector space, and therefore is closed under addition of elements and multiplication by scalars, it is easy to stay in $\mathfrak{so}(n)$. In this way we can make sure that we always stay on $SO(n)$.

We can also give a slightly simpler form of this method, once we realize that $\mathbf{W}' = \mathbf{R}\mathbf{W}$ for some $\mathbf{R} \in SO(n)$. Moving from \mathbf{W} to \mathbf{W}' is equivalent to moving from \mathbf{I}_n to \mathbf{R} , and we already know that $\log \mathbf{I}_n = \mathbf{0}_n$. Our alternative method is therefore

- (1) Start at $\mathbf{0}_n \in \mathfrak{so}(n)$, equivalent to $\mathbf{I}_n \in SO(n) = \exp(\mathbf{0}_n)$
- (2) Move about in $\mathfrak{so}(n)$ from $\mathbf{0}_n$ to $\mathbf{B} \in \mathfrak{so}(n)$
- (3) Use $\exp(\cdot)$ to map back into $SO(n)$, giving $\mathbf{R} = \exp(\mathbf{B})$
- (4) Calculate $\mathbf{W}' = \mathbf{R}\mathbf{W} = \exp(\mathbf{B})\mathbf{W} \in SO(n)$.

This latter method avoids having to calculate the logarithm.

7.3 Choosing the search direction: Steepest gradient descent

The intuitively natural search direction to choose is the “steepest descent” direction that we have discussed before. However, we have seen that this depends entirely by what we mean by “distance”. So far we have not included

any concept of distance on our manifold, and many results about manifolds can follow without one [18], but we will now introduce one to help us choose our search direction.

The Lie algebra $\mathfrak{so}(n)$ is the set of skew-symmetric matrices, so it is a vector space with $n(n-1)/2$ dimensions, corresponding to the entries in the upper triangle of any matrix $\mathbf{B} \in \mathfrak{so}(n)$. There seems no a-priori reason to favour any component over any other, and there seem to be no “special” directions to choose, so a simple “length” measure could be that given by $l_{\mathbf{B}}^2 \equiv |\mathbf{B}|^2 = \frac{1}{2} \|\mathbf{B}\|_F^2 = \sum_{ij} b_{ij}^2/2$. The factor of two is included to reflect the fact that each component is repeated (in the negative) in the other half of the matrix. This also suggests an inner product (or “dot product”) $\langle \mathbf{B}, \mathbf{H} \rangle = \sum_{ij} b_{ij} h_{ij}/2$ so that $l_{\mathbf{B}}^2 = \langle \mathbf{B}, \mathbf{B} \rangle$. (A vector space with an inner product like this, that gives a norm that behaves like a distance metric, is called a *Hilbert space*.) With this length measure, equal-length elements of $\mathfrak{so}(n)$ are those with equal Frobenius norm $\|\mathbf{B}\|_F$.

Now we consider the gradient in this \mathbf{B} -space, our Lie algebra $\mathfrak{so}(n)$. In the original unconstrained \mathbf{W} -space we used the gradient $\nabla_{\mathbf{W}} J$ and the matrix of partial derivatives $\partial J / \partial \mathbf{W} = [\partial J / \partial w_{ij}]$ as if they were synonymous. However, for \mathbf{B} we have to be a little more careful, since the components of \mathbf{B} are not all independent, and the length and inner product is not quite the same as the implicit length and inner product we would use for normal matrices.

For our purposes, we will define the gradient $\nabla_{\mathbf{B}} J$ in words as follows: *The gradient $\nabla_{\mathbf{B}} J$ of J in \mathbf{B} -space is a vector whose inner product $\langle \nabla_{\mathbf{B}} J, \mathbf{H} \rangle$ with a unit vector \mathbf{H} in \mathbf{B} -space is equal to the component of the derivative of $J(\mathbf{B})$ in the direction \mathbf{H} .* This definition is intimately tied to the inner product $\langle \cdot, \cdot \rangle$

which we are using for our \mathbf{B} -space, and hence to the distance metric generated by the inner product. Since our \mathbf{B} -space, $\mathfrak{so}(n)$, is a space of skew-symmetric matrices, the gradient (and the unit vector \mathbf{H} in this definition) will also be skew-symmetric.

Suppose we have the derivatives in \mathbf{W} -space $\partial J/\partial \mathbf{W}$, and let $\mathbf{W} = \exp(\mathbf{B})\mathbf{W}_0$ and let \mathbf{B} vary in direction \mathbf{H} as $\mathbf{B} = t\mathbf{H}$, where \mathbf{H} is a unit vector, so that $\mathbf{W} = \exp(t\mathbf{H})\mathbf{W}_0$. Then we have $d\mathbf{W}/dt = \mathbf{H}\exp(t\mathbf{H})\mathbf{W}_0 = \mathbf{H}\mathbf{W}$, and the derivative of $J(\mathbf{B})$ along the line $\mathbf{B} = t\mathbf{H}$ is

$$\begin{aligned} dJ/dt &= \text{trace}((\partial J/\partial \mathbf{W})^T (d\mathbf{W}/dt)) = \text{trace}((\partial J/\partial \mathbf{W})^T \mathbf{H}\mathbf{W}) \\ &= \text{trace}(\mathbf{W}(\partial J/\partial \mathbf{W})^T \mathbf{H}) = \text{trace}(\text{skew}(\mathbf{W}(\partial J/\partial \mathbf{W})^T) \mathbf{H}) \end{aligned}$$

where the last equality follows since \mathbf{H} is skew-symmetric. By our definition of the inner product for \mathbf{B} we have $\langle \nabla_{\mathbf{B}} J, \mathbf{H} \rangle = \frac{1}{2} \text{trace}((\nabla_{\mathbf{B}} J)^T \mathbf{H}) = \text{trace}((\frac{1}{2} \nabla_{\mathbf{B}} J)^T \mathbf{H})$ which by definition of the gradient must equal dJ/dt , so we have

$$\nabla_{\mathbf{B}} J = 2 \text{skew}((\partial J/\partial \mathbf{W}) \mathbf{W}^T) = (\partial J/\partial \mathbf{W}) \mathbf{W}^T - \mathbf{W}(\partial J/\partial \mathbf{W})^T \quad (45)$$

(ensuring that $\nabla_{\mathbf{B}} J$ is skew-symmetric) or alternatively $\nabla_{\mathbf{B}} J = (\nabla_{\mathbf{W}} J) \mathbf{W}^T - \mathbf{W}(\nabla_{\mathbf{W}} J)^T$.

For the non-negative ICA task substituting (6) into (45) with $\nabla_{\mathbf{W}} J = \partial J/\partial \mathbf{W}$ we get

$$\nabla_{\mathbf{B}} J = 2 \text{skew}(E(\mathbf{y}_- \mathbf{z}^T) \mathbf{W}^T) = E(\mathbf{y}_- \mathbf{y}^T) - E(\mathbf{y} \mathbf{y}_-^T) \quad (46)$$

which is of a particularly simple form and is *equivariant*, since it a function of the output \mathbf{y} only [5].

7.4 Steepest descent in $\mathfrak{so}(n)$: geodesic flow

A simple update method is to update \mathbf{W} using the method outline in section 7.2, by making a small step in $\mathfrak{so}(n)$ from $\mathbf{B} = \mathbf{0}$ to $\mathbf{B} = -\eta \nabla_{\mathbf{B}} J|_{\mathbf{B}=\mathbf{0}}$, in turn moving in $\text{SO}(n)$ from \mathbf{W}_k to

$$\mathbf{W}_{k+1} = \exp(-\eta \mathbf{G}) \mathbf{W}_k \quad (47)$$

where

$$\mathbf{G} = \nabla_{\mathbf{B}} J|_{\mathbf{B}=\mathbf{0}} = (\nabla_{\mathbf{W}} J) \mathbf{W}^T - \mathbf{W} (\nabla_{\mathbf{W}} J)^T. \quad (48)$$

which is the *geodesic flow* method introduced to ICA by Nishimori [19]. Since \mathbf{G} is skew-symmetric, we have $\exp(-\eta \mathbf{G}) \in \text{SO}(n)$, so \mathbf{W} remains in $\text{SO}(n)$ provided it was initially, for example by setting $\mathbf{W}_0 = \mathbf{I}_n$.

Geodesic flow is the equivalent on a constrained manifold to the steepest descent procedure $\mathbf{W}_{k+1} = \mathbf{W}_k - \eta \nabla_{\mathbf{W}} J$ in the usual Euclidean space with the usual Euclidean distance measure. The idea of a *geodesic*, the shortest path between two points on a manifold, is the generalization of the concept of a *straight line* in normal Euclidean space. However, note that the “geodesic flow” method only *locally* follows the geodesic: as the gradient $\nabla_{\mathbf{W}} J$ (and $\nabla_{\mathbf{B}} J$) changes, the path traced out will most likely be a more general curve, much as a normal steepest descent in Euclidean space only *locally* follows a straight line.

This geodesic flow (47) for small η approximately follows the flow of the ordinary differential equation $d\mathbf{W}/dt = \nabla_{\mathbf{W}} J$ where $\nabla_{\mathbf{W}}$ is restricted to be in the tangent space $T_{\mathbf{W}}$ at \mathbf{W} . In section 4.3 we examined this constraint in \mathbf{W} space, but saw that a finite update to \mathbf{W} would eventually “drift” away from the surface of the manifold. For geodesic flow it turns out that the cal-

culated gradient is identical, but the use of the Lie algebra with the mapping $\exp(-\eta\mathbf{G})$ to Lie group ensures that \mathbf{W} is always constrained to lie on the manifold corresponding to the desired constraint. In real numerical simulations, the accuracy of this approach this will depend on the calculation of the matrix exponential $\exp(\cdot)$, and some care needs to be taken here [20].

For the non-negative ICA task we use the geodesic flow equation (47) with $\mathbf{G} = \nabla_{\mathbf{B}}J = E(\mathbf{y}_-\mathbf{y}^T) - E(\mathbf{y}\mathbf{y}_-^T)$ from (46), where \mathbf{y} and \mathbf{y}_- are measured at $\mathbf{B} = 0$ [21].

8 Geodesic search over one-parameter subgroups

Suppose now that we want to make larger movements, or several movements in $\mathfrak{so}(n)$. The Lie bracket over $\mathfrak{so}(n)$ does make life a little awkward here: it makes it tricky to work out *how* we should move about in the Lie algebra if we want to take more than one small step at a time. To be specific, we have the following fact [22]

$$\exp(\mathbf{A})\exp(\mathbf{B}) = \exp(\mathbf{B})\exp(\mathbf{A}) = \exp(\mathbf{A} + \mathbf{B}) \quad \text{if and only if} \quad [\mathbf{A}, \mathbf{B}] = 0. \quad (49)$$

In other words, we can only use addition in the Lie algebra in the way that we are used to, where addition in the Lie algebra $\mathfrak{so}(n)$ corresponds to multiplication in the Lie group $\text{SO}(n)$, when we have a pair of matrices that commute (for example, if we have an Abelian Lie algebra).

8.1 Searching along one-parameter subalgebras

A simple way to allow us to make a single step in the Lie algebra is always to start from the zero point in our Lie algebra, since the matrix \mathbf{O}_n and its exponential $\mathbf{I}_n = \exp \mathbf{O}_n$ commute with any matrix. Therefore we can move in any direction $\mathbf{B} = t\mathbf{H}$ in our Lie algebra, for real scalar t and skew-symmetric matrix \mathbf{H} and be sure that all steps in this direction will commute. The set of matrices $\mathfrak{g}_{\mathbf{H}} = \{t\mathbf{H} | t \in \mathbb{R}\}$ is a Lie algebra itself, called a *one-parameter subalgebra* of $\mathfrak{so}(n)$: every element of this subalgebra is $t\mathbf{H}$ for a particular value of the one parameter t . All pairs of matrices of the form $t_1\mathbf{H}$ and $t_2\mathbf{H}$ commute, which from (49) implies that their exponentials $\mathbf{R}_1 = \exp(t_1\mathbf{H})$ and $\mathbf{R}_2 = \exp(t_2\mathbf{H})$ commute, so $\mathfrak{g}_{\mathbf{H}}$ is an Abelian (commutative) algebra with a corresponding *one-parameter subgroup* of matrices $\mathbf{R}(t) = \exp(t\mathbf{H})$.

Therefore by choosing the “direction” \mathbf{H} , we can choose to take steps along this “line” (in this subalgebra) until we have sufficiently reduced our cost function J . This is a generalization of the idea of a “straight line” that we would use as a search direction in a more usual Euclidean search space. In Euclidean space \mathbb{R}^n , starting from a point $\mathbf{w}(0)$, we would choose a search direction \mathbf{h} and search for a point $\mathbf{w}(t) = \mathbf{w}(0) + t\mathbf{h}$ which minimises our cost function $J(\mathbf{w})$. In our new formulation, our approach is as follows:

- (1) Our starting point for is $\mathbf{W}_k(0) \in \text{SO}(n)$, which we write as $\mathbf{R}(0)\mathbf{W}(0)$ where $\mathbf{R}(0) = \mathbf{I}_n \in \text{SO}(n)$
- (2) We choose a search direction \mathbf{H} in our Lie algebra $\mathfrak{so}(n)$
- (3) We search along the points in the one parameter subalgebra $t\mathbf{H}$, corresponding to points in the one-parameter subgroup $\mathbf{W}_k(t) = \mathbf{R}(t)\mathbf{W}_k(0)$

where $\mathbf{R}(t) = \exp(t\mathbf{H})$, in order to reduce $J(\mathbf{R}(t)\mathbf{W}_k(0))$

- (4) Once we have reduced J far enough, we update $\mathbf{W}_{k+1}(t) = \mathbf{R}(t)\mathbf{W}_k(0)$ and start again with a new “line” search.

We are now left with two choices to make: (1) how do we choose the search direction, and (2) the method to use to search along the “line” (the one-parameter subgroup/subalgebra).

8.2 Line search and geodesic search

The correspondence between geodesic flow and steepest descent search leads to the idea of generalizing a Euclidean “steepest descent line search” method into a *geodesic search* method. We use the steepest descent direction in our Lie algebra (\mathbf{B} -space) to determine a one-parameter geodesic in our \mathbf{W} -space, then make large steps along that geodesic until we find the minimum [21]. This leads to the following method:

- (1) Calculate $\mathbf{G} = \nabla_{\mathbf{B}}J$ and $\theta = |\mathbf{G}|$. If θ is small or zero exit, otherwise calculate $\mathbf{H} = -\mathbf{G}/\theta$.
- (2) Search along $\mathbf{R}(t) = \exp(t\mathbf{H})$ using $\mathbf{y} = \mathbf{R}\mathbf{W}\mathbf{z}$ to find a minimum (or near-minimum) of J at $t = t^*$.
- (3) Update $\mathbf{W}_{k+1} = \mathbf{R}(t^*)\mathbf{W}_k$
- (4) Repeat from step 1 until θ is small enough to exit.

This framework can have several variations, as does the usual line search method. For example, if we simply use the update $t = \eta\theta$ instead of a complete search at step 2, we have the geodesic flow algorithm with $\mathbf{R} = \exp(-\eta\theta\mathbf{H}) = \exp(-\eta\mathbf{G})$. If we “think” in our single-parameter space $t \in \mathbb{R}$, there are many

possible line search methods which can be used [15], since this is conceptually identical to a standard line search.

8.3 Newton update step

If J is twice differentiable with respect to t , we can also consider a Newton method along this search line. For this we need to calculate dJ/dt and d^2J/dt^2 for t in the geodesic search step 2 above, once we have chosen \mathbf{H} . Since \mathbf{H} was chosen to be in the direction of the negative gradient, we immediately have $dJ/dt = \sum_{ij} [\mathbf{H}]_{ij} [\nabla_{\mathbf{B}} J]_{ij} = -\sum_{ij} [\nabla_{\mathbf{B}} J]_{ij}^2 / \theta = -2\theta$ where $\theta = |\nabla_{\mathbf{B}} J|$ as defined above. The evaluation of d^2J/dt^2 could be carried out from the $n^2 \times n^2$ Hessian matrix of $d^2J/dw_{ij}dw_{kl}$, but it can often be evaluated directly, avoiding calculating the $n^2 \times n^2$ Hessian matrix.

The Newton line step is then simply to set $t = -(dJ/dt)/(d^2J/dt^2)$. Note that the unmodified Newton method will not always find a minimum. For example, if the curvature d^2J/dt^2 is negative, the Newton method will step “upwards” towards a maximum, rather than downwards towards a minimum, so we may wish to modify the Newton method to be more like steepest descent in this case [15].

For the non-negative ICA task evaluation of dJ/dt is straightforward given $\nabla_{\mathbf{B}} J$, and further differentiation of d^2J/dt^2 yields [16]

$$d^2J/dt^2 = E(|\mathbf{k}_- \circ (\mathbf{H}\mathbf{y})|_2^2 + \mathbf{y}_-^T \mathbf{H}^2 \mathbf{y}) \quad (50)$$

where \mathbf{k}_- is an indicator vector such that $(k_-)_i = 1$ if $y_i < 0$ and zero otherwise, and \circ represents element-wise multiplication. Equation (50) appears in

a batch version in [16].

8.4 Fourier update step

For the special case of $\text{SO}(3)$, i.e. $n = 3$ inputs, our one-parameter subgroup takes on a special form. Each choice of $\mathbf{H} \in \mathfrak{so}(3)$ defines a rotation about a particular axis through the origin, so the subspace of rotation can be defined by its “dual” subspace consisting of the axis about which the rotation occurs. We are very used to this idea of rotations about an axis in our usual 3-dimensional world. The exponential of a skew-symmetric matrix also has a special form for $n = 3$, the Rodrigues formula [23,24]:

$$\exp(\mathbf{B}) = \mathbf{I}_n + (\sin \phi)\mathbf{B}_1 + (1 - \cos \phi)\mathbf{B}_1^2 \quad (51)$$

where $\phi = |\mathbf{B}| = \|\mathbf{B}\|_F/\sqrt{2}$ and $\mathbf{B}_1 = \mathbf{B}/\phi$, provided that $\phi \neq 0$.

Now it is clear from equation (51) that $\exp(\mathbf{B})$ repeats every $\phi = 2\pi$, as for the $n = 2$ case. Therefore for $n = 3$, *once we have chosen a search direction*, we also know that we are performing a rotation search. This means that we can if we wish also use a Fourier expansion of J to perform this line search as in section 6.4 above [16]. We might therefore argue that this search is “easier” than the usual line search to minimize J , since we immediately know that the minimum (or at least a copy of the minimum) must lie in the range $0 \leq t < 2\pi$. The first-order Fourier expansion requires dJ/dt and d^2J/dt^2 , as for the Newton method, and approximates the Newton method for small updates.

9 Conjugate Gradients methods

Now that we have seen that we can perform a steepest descent line search to minimize J , it is natural to ask if we can also use the well-know ‘‘Conjugate Gradients’’ method. Edelman, Arias and Smith [25] constructed conjugate gradient algorithms on similar manifolds, and this approach was used by Martin-Clemente et al [26] for ICA.

In the most familiar forms, the standard conjugate gradient algorithm, in a Euclidean search space, calculates a new search direction based on two previous search directions [15] (for a particularly clear introduction to conjugate gradient methods with plenty of visualization, see [27]). For example, we have the steps

- (1) Set $k = 0$ and choose an initial search direction $\mathbf{h}_0 = -\mathbf{g}_0$ where $\mathbf{g}_0 = \nabla J$
- (2) Choose $t_k = t_k^*$ to minimize $J(\mathbf{x})$ along the line $\mathbf{x} = \mathbf{x}_k + t_k \mathbf{h}_k$
- (3) At the minimum, find the new gradient $\mathbf{g}_{k+1} = \nabla_{\mathbf{x}} J$ at t_k^*
- (4) Choose a new search direction $\mathbf{h}_{k+1} = -\mathbf{g}_{k+1} + \gamma_k \mathbf{h}_k$ where

$$\gamma_k = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} \quad \text{or} \quad \gamma_k = \frac{\mathbf{g}_{k+1}^T (\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^T \mathbf{g}_k} \quad (52)$$

- (5) Repeat from step 2 with $k = k + 1$ until the gradient \mathbf{g}_k is sufficiently small

The first equation for γ_k here is the Fletcher-Reeves formula, the second is the Polak-Ribière formula. They are both equal for quadratic functions and perfect minimization, but the Polak-Ribière formula is generally considered to perform better for nonlinear problems.

9.1 Parallel transport

The difficulty with implementing this on a manifold is that, while the derivatives and gradients at all points in a Euclidean space “live” in the same tangent space, the derivatives at each point \mathbf{W} on a general manifold live in a tangent space $T_{\mathbf{W}}$ at \mathbf{W} , which is specific to the particular point \mathbf{W} . To manipulate gradients \mathbf{g} and search directions \mathbf{h} , each of which are derivative-like objects which “live” in the tangent space, we must first move them to the same point on the manifold. In a space with an inner product and norm defined, as we have done for our $\mathfrak{so}(n)$ (and hence $\text{SO}(n)$) system, we can use the idea of *parallel transport* along a geodesic to do this [25]. (The inner product and the geodesic curve help to define a *connection* between the different tangent spaces $T_{\mathbf{W}(t)}$, to allow us to map a derivative in one tangent space $T_{\mathbf{W}(t_1)}$ at $\mathbf{W}(t_1)$ to another tangent space $T_{\mathbf{W}(t_2)}$ at $\mathbf{W}(t_2)$).

Suppose we have a gradient vector \mathbf{g}_k at the point \mathbf{W}_k on our manifold: this gradient vector must live in the tangent space $T_{\mathbf{W}_k}$ at \mathbf{W}_k . The idea of parallel transport is that we move (*transport*) this along the curve $\mathbf{W}(t) = \mathbf{R}(t)\mathbf{W}_k$ as t changes from 0 to t^* . In doing so we move it to the corresponding vector in each tangent space $T_{\mathbf{W}(t)}$, keeping it pointing in the corresponding direction in each tangent space.

Imagine doing this on a globe, where our tangent spaces are flat planes. As we move along $\mathbf{W}(t)$, we carry the plane with us, keeping it aligned with the geodesic we are traveling along (Fig. 7). When we reach our destination $\mathbf{W}(t^*)$, the direction on the plane corresponding to \mathbf{g}_k at \mathbf{W}_k is now the parallel-transported version $\tau\mathbf{g}_k$ of \mathbf{g}_k , along the curve $\mathbf{W}(t) = \mathbf{R}(t)\mathbf{W}_k$, at

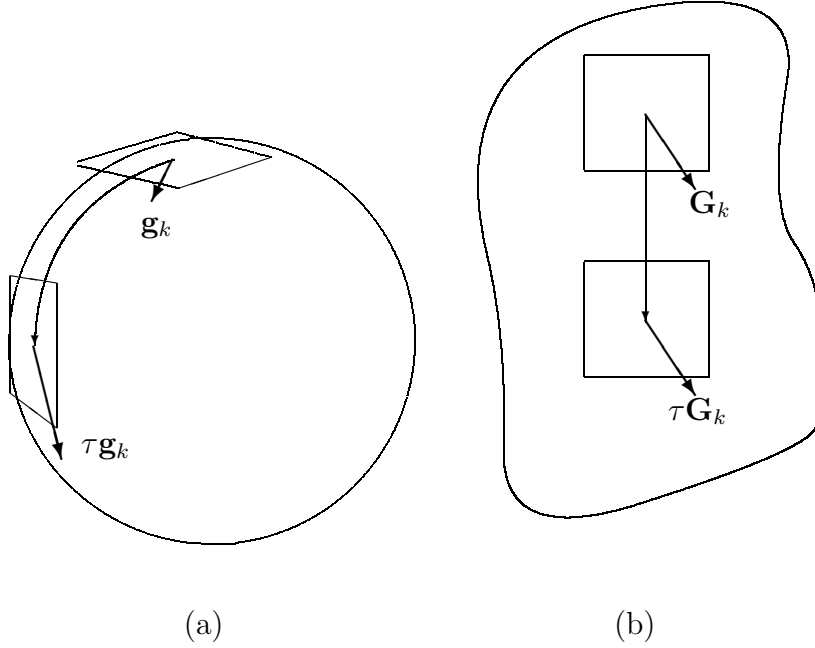


Fig. 7. Parallel transport of \mathbf{g}_k to $\tau\mathbf{g}_k$, in (a) the Lie group and (b) the Lie algebra

$\mathbf{W}_{k+1} = \mathbf{R}(t^*)\mathbf{W}_k$. Notice that this idea of parallel transport depends critically on the geodesic we follow: if we followed a different one to get from \mathbf{W}_k to \mathbf{W}_{k+1} , we could easily end up with a different parallel-transported version $\tau'\mathbf{g}_k$ of \mathbf{g} [18].

It can be easier to think of this in the Lie algebra $\mathfrak{so}(n)$ (Fig. 7(b)). If we work in the algebra $\mathfrak{so}(n)$ and move along geodesics (“straight lines”), i.e. move within one-parameter subalgebras of the form $\mathbf{B} = t\mathbf{H}$ for skew-symmetric \mathbf{H} , then the parallel transported version of a gradient vector $\mathbf{G} \in \mathfrak{so}(n)$ at from $\mathbf{B} = t\mathbf{H}$ to $\mathbf{B}' = t'\mathbf{H}$ is simply \mathbf{G} itself. Consequently if we work in the subalgebra \mathbf{B} we can effectively ignore the problem of parallel transport, and use the conjugate gradient method with the update equation (52) directly.

9.2 Conjugate gradients with second order information

The Fletcher-Reeves or Polak-Ribière formulae are finite difference approximations to an underlying principle: the principle that the search direction \mathbf{h}_{k+1} should be chosen to be *conjugate* to the search direction \mathbf{h}_k . Therefore if $\nabla^2 J$ is the Hessian matrix of the system, $[\nabla^2 J]_{ij} = d^2 J/dx_i dx_j$ then the direction \mathbf{h}_{k+1} should satisfy the following requirement:

$$\mathbf{h}_{k+1}^T (\nabla^2 J) \mathbf{h}_k = 0. \quad (53)$$

We would like to avoid calculating the Hessian $\nabla^2 J$ if we can. We can do this if we notice that $d/dt_k(\nabla J) = (\nabla^2 J)\mathbf{h}_k$ is a vector only, reflecting the change in gradient ∇J as we move along the line $\mathbf{x} = \mathbf{x}_k + t\mathbf{h}_k$. Therefore we only need to calculate the projection of the Hessian onto the line \mathbf{h}_k , rather than the full Hessian.

In our Lie algebra setting, we need to calculate $d/dt(\nabla_{\mathbf{B}} J)$ for $\mathbf{B} = t\mathbf{H}$ where \mathbf{H} is the search line we have “just come along”. Let us denote this as

$$\mathbf{V}_k = \frac{d}{dt} \nabla_{\mathbf{B}} J = 2 \text{skew} \left(\left(\frac{d}{dt} \nabla_{\mathbf{W}} J \right) \mathbf{W}^T \right) \quad (54)$$

where the second equality comes from (45). Note that, as Edelman et al [25] point out, the conjugacy here does not simply use the Hessian matrix $\nabla_{\mathbf{W}}^2 J$ in \mathbf{W} -space, but instead uses the Hessian in \mathbf{B} -space, our Lie algebra.

Given (54) we can choose the direction \mathbf{H}_{k+1} to be the component of \mathbf{G}_{k+1} which is orthogonal to $\mathbf{V}_k = \frac{d}{dt} \nabla_{\mathbf{B}} J$ at step k , by subtracting the component in that direction. Writing

$$\tilde{\mathbf{G}}_{k+1} = \mathbf{V}_k \frac{\langle \mathbf{G}_{k+1}, \mathbf{V}_k \rangle}{\langle \mathbf{V}_k, \mathbf{V}_k \rangle} \quad (55)$$

as the component of \mathbf{G}_{k+1} in the direction of \mathbf{V}_k , we set the new search direction to be $\mathbf{H}_{k+1} = \tilde{\mathbf{H}}_{k+1}/|\tilde{\mathbf{H}}_{k+1}|$ where $\tilde{\mathbf{H}}_{k+1} = \mathbf{G}_{k+1} - \tilde{\mathbf{G}}_{k+1}$. It is easy to verify that this update ensures $\langle \mathbf{H}_{k+1}, \mathbf{V}_k \rangle = 0$. Using the formula (54) for $\frac{d}{dt} \nabla_{\mathbf{B}} J$ we therefore have a conjugate gradient algorithm using second order derivative information instead of finite difference information, but which still avoids calculating the complete Hessian matrix.

For the non-negative ICA task a little manipulation gives us the projection of the Hessian along the line $\mathbf{B} = t\mathbf{H}$ as

$$\mathbf{V}_k = \frac{d}{dt} \nabla_{\mathbf{B}} J = -E(\text{skew}(\dot{\mathbf{y}}_- \mathbf{y}^T + \mathbf{y}_- \dot{\mathbf{y}}^T)) \quad (56)$$

where $\dot{\mathbf{y}} = \frac{d}{dt} \mathbf{y} = -\mathbf{H}\mathbf{y}$ and $[\dot{\mathbf{y}}_-]_{ij} = [\dot{\mathbf{y}}]_{ij}$ if $y_{ij} < 0$ or zero otherwise.

10 Exponentiation and Toral Subalgebras

There are still some issues that remain to be considered. One is the calculation of the exponent: every time we map from an element $t\mathbf{H} \in \mathfrak{so}(n)$ in the Lie algebra, so that we can evaluate the cost function J or its derivatives, we need to calculate the matrix exponential $\exp(t\mathbf{H})$. Many ways have been proposed to do this, such as the Padé approximation used by Matlab, and there are methods specifically for $\mathfrak{so}(n)$ based on the Rodrigues formula that we have encountered above, and its generalizations [20,23,24].

Matrix exponentiation is computationally expensive, and many methods are still being proposed to approximate this efficiently for Lie group applications [28–30]. An alternative approach is to use the bilinear *Cayley transform*, approximating $\exp(\mathbf{B})$ by $\text{cay}(\mathbf{B}) \equiv (\mathbf{I}_n + \mathbf{B}/2)(\mathbf{I}_n - \mathbf{B}/2)^{-1} = (\mathbf{I}_n - \mathbf{B}/2)^{-1}(\mathbf{I}_n +$

$\mathbf{B}/2$) [17,31–33] which has been used in place of matrix exponentiation as a map to generate orthogonal matrices [34]. However, the Cayley transform does give us a different effective coordinate system to the Lie algebra we have discussed so far, and we will not consider it further in this article.

We can also consider how we can reduce the *number* of exponentials we need to calculate. In particular, our task is to optimize a particular function, not to follow a particular ode: therefore we do not necessarily need to choose a new “steepest descent” or “conjugate gradient” search direction \mathbf{H} if a more sensible choice would be appropriate. Let us introduce the *Jordan canonical form* of a skew-symmetric matrix: this will allow us to see how to decompose a movement on $\text{SO}(n)$ into commutative parallel rotations.

10.1 Visualizing $\text{SO}(n)$ for $n \geq 4$: the *Jordan canonical form*

The case of $\text{SO}(n)$ for $n \geq 4$ is not quite as simple as for the cases $n = 2$ or $n = 3$, which we saw are equivalent to circular rotations about an axis (for $n = 3$) or about the origin of the plane (for $n = 2$). For $n = 2$, exponentiation is a simple matrix involving sin and cos terms, while for $n = 3$ we can use the Rodrigues formula (51).

However, our insight for $n \geq 4$ is rescued by the *Jordan canonical form* for skew-symmetric matrices. Any skew-symmetric matrix \mathbf{B} can be written in

$\mathbf{R}_1 \mathbf{R}_2 \cdots \mathbf{R}_m$ where each of the matrices \mathbf{R}_i are of the form

$$\mathbf{R}_i = \exp \mathbf{B}_i = \mathbf{U} \begin{pmatrix} \mathbf{I}_{2(i-1)} & & \\ & \mathbf{M}_i & \\ & & \mathbf{I}_{n-2(i+1)} \end{pmatrix} \mathbf{U}^T. \quad (59)$$

To be specific, for $n = 4$ this means that a search direction \mathbf{H} corresponds, in general, to a pair of commutative rotations in orthogonal 2-dimensional planes.

Interestingly, since \mathbf{B} is skew-symmetric, we can use a *symmetric* eigenvalue decomposition method to find its eigenvectors, as follows [24]. Since $\mathbf{B}^T = -\mathbf{B}$, and therefore $\mathbf{B}^T \mathbf{B} = -\mathbf{B}^2$, this means $-\mathbf{B}^2$ is a symmetric positive semi-definite matrix. Given also that $\mathbf{B} \mathbf{B}^2 = \mathbf{B}^2 \mathbf{B}$ this commutes with \mathbf{B} so the eigenvectors of $\mathbf{B}^T \mathbf{B}$ are identical to those of \mathbf{B} . The eigenvalues of $\mathbf{B}^T \mathbf{B}$ occur in pairs, where each pair corresponds to one block Θ_i in the Jordan canonical form of \mathbf{B} [24].

If the canonical form contains just a single 2×2 block, this would correspond to a single rotation plane, and we would return to the starting point after $\theta = 2\pi$ as for the $n = 2$ and $n = 3$ case. However, if \mathbf{H} is chosen at random, we *never* return to the starting point in $\text{SO}(4)$. To do so at some parameter t would require $t\theta_1 = 2k_1\pi$ and $t\theta_2 = 2k_2\pi$ for some integers k_1 and k_2 - i.e. we must have complete a whole number of revolutions in *both* rotation planes. This can only happen if θ_1/θ_2 is rational, which will never be the case (at least, it has zero probability) for randomly chosen \mathbf{H} . At the very least, on a computer with finite precision arithmetic, the ratio θ_1/θ_2 may involve very large numbers, so it may take a long time to return exactly to our starting

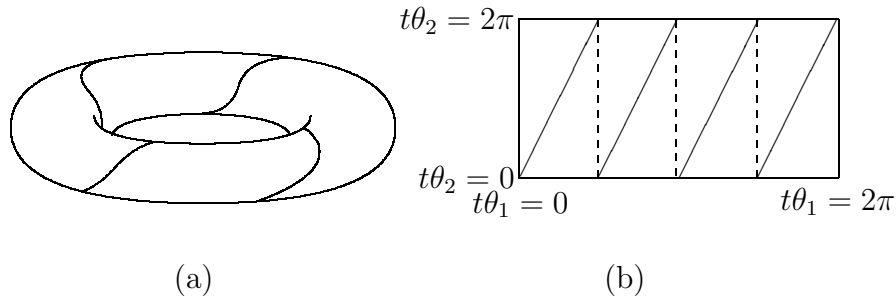


Fig. 8. One-parameter subgroup on a torus (a). In this case we have $t\theta_2 = 4t\theta_1$ so we rotate four times “around” while rotating one time “along”. It can often help to visualize this flattened out (b).

point.

This generalizes naturally for $\text{SO}(n)$ with $n > 4$, either even $n = 2p$ or odd $n = 2p + 1$. The Jordan canonical form tells us that, given a fixed $n \times n$ skew-symmetric matrix \mathbf{H} , moving in a one-parameter subgroup $\exp(t\mathbf{H})$ within in $\text{SO}(n)$ causes a rotation in p orthogonal planes, and the rotation in these planes is commutative. We are actually moving on the surface of a p -dimensional *torus*, where $p = 1$ gives us a circle, and $p = 2$ gives the familiar doughnut shaped torus we are used to. The one-parameter subgroup forms a “spiral” around the torus. If the parameters θ_i in our Jordan canonical form are rationally related, the spiral will meet itself again after “going round” an appropriate number of times (Fig. 8).

10.2 A commutative, toral subalgebra

We have seen that decomposing \mathbf{B} into Jordan canonical form yields a set of m commutative matrices \mathbf{B}_i , where $n = 2m$ or $2m + 1$, such that $\mathbf{B} = \sum_i \mathbf{B}_i$ and $\mathbf{R} = \exp(\mathbf{B}) = \mathbf{R}_1 \cdots \mathbf{R}_m$. Since these are commutative, we can make a number of “movements” within this decomposition, where each will have a

simple exponential, given that the decomposition already exists.

Given our decomposition set $\{\mathbf{B}_i\}$, suppose we fix a basis $\mathbf{H}_i = \mathbf{B}_i/|\mathbf{B}_i|$ and set t_i such that $\mathbf{B}_i = t_i\mathbf{H}_i$. Then each of the coordinates t_i determines an angle of rotation θ_i around a circle. In fact the set of coordinates $\{t_i\}$ trace out a *toral subalgebra* $\mathfrak{t} \subset \mathfrak{so}(n)$. If the number of components m is maximal, so that for $\text{SO}(n)$ we have $n = 2m$ or $2m + 1$ (depending on whether n is even or odd), this will correspond to a *maximal torus* of $\text{SO}(n)$. We therefore have an m -dimensional subgroup/subalgebra to “move about” on, which is half (or just under half) the dimensionality of the original n -dimensional manifold. Unlike the original n -dimensional manifold, however, this subalgebra is Abelian, so movements on the corresponding torus are commutative. This means we can make the movements in any order we wish, justifying the use of the “plus” notation for $\mathbf{B} = t_1\mathbf{H}_1 + \cdots + t_m\mathbf{H}_m$.

We only have to perform the decomposition into canonical form once each time we choose the toral subalgebra. Each movement to a point \mathbf{B} in the subalgebra then requires m calculations of $\sin \theta$ and $\cos \theta$ (as well as the usual matrix manipulations) to map to the corresponding point $\mathbf{W} \in \text{SO}(n)$ on the manifold, since the orthogonal matrix \mathbf{U} used in the decomposition (57) is fixed. To search on the subalgebra we could use the conjugate gradient method, for example, restricted to the toral subalgebra $\mathfrak{t} \subset \mathfrak{so}(n)$. Only once we have finished searching in the subalgebra do we need to calculate a new decomposition, perhaps based on a new unrestricted steepest-descent direction.

Perhaps the reason this concept seems a little strange at first is that the first nontrivial example does not appear until $n = 4$. The group $\text{SO}(2)$ is itself isomorphic to (behaves the same as) the circle S^1 (the *1-torus*) so $\text{SO}(2)$ is

essentially the same as its own maximal torus. The group $\text{SO}(3)$ also has the circle S^1 as its maximal “torus”. This is the one-parameter geodesic that we have used before, that rotates about a fixed axis. Only when we reach $\text{SO}(4)$ do we have the maximal torus $(S^1)^2$, giving us a two-dimensional maximal torus (corresponding to the familiar doughnut shape) to search on between decompositions. This corresponds to simultaneous, and commutative, rotations in two orthogonal planes in our four-dimensional space.

10.3 Component-aligned toral subalgebra

We have so far considered that we generate our torus using, for example, an initial steepest-descent direction \mathbf{H} , for which we perform decomposition into Jordan canonical form. This ensures that the generated torus will contain \mathbf{H} . However, this is only one way to generate the torus from \mathbf{H} : we could instead construct a torus based on the components h_{ij} of \mathbf{H} , where our “decomposition” requires only permutations of the rows and columns. Specifically, given some generator matrix $\mathbf{H} \in \mathfrak{so}(n)$, we could build a torus from this, using the following algorithm:

- (1) Initialize $k = 1$, $\mathbf{U} = \mathbf{0}$
- (2) Find $(i^*, j^*) = \arg \max_{(i,j)} h_{ij}$
- (3) Set (row, col) element $(i^*, 2k - 1)$ of \mathbf{U} to 1 and element $(j^*, 2k)$ of \mathbf{U} to 1
- (4) Set all elements in row i^* and column j^* of \mathbf{H} to zero
- (5) If $k < m$ repeat from step 2 with $k \leftarrow k + 1$.

This results in a matrix \mathbf{U} for the decomposition (57) which is a permutation matrix, forming our torus from m pairs of different axes to form the planes of rotation, with no axis used more than once. While this is simpler than the torus generated by an eigenvector-based decomposition of \mathbf{H} , this component-aligned torus may no longer contain \mathbf{H} itself, so it may be that the minimum of J on this torus is not as good as it would have been had we generated our torus as in the previous method which included \mathbf{H} . Some experimentation will be required to evaluate whether the cost saving in evaluating the decomposition for $\exp(\mathbf{H})$ is worth the extra minimization steps that may be required.

10.4 Conjecture: a possible torus-based replacement for Newton

So far we have assumed that we could use an inner product and a distance metric to find a “steepest descent” direction to minimize J . To do so we defined an inner product $\langle \mathbf{B}, \mathbf{H} \rangle = \frac{1}{2} \text{trace}(\mathbf{B}^T \mathbf{H})$ and a length $|\mathbf{B}| = \langle \mathbf{B}, \mathbf{B} \rangle^{1/2}$ in our Lie algebra $\mathfrak{so}(n)$, arguing that this was the “natural” distance metric since all the components in \mathbf{B} are similar to each other.

However, the concept of the toral subalgebra suggests that the use of a 2-norm metric like this may not be the necessary, or perhaps even desirable. The fact that if we “cut open” a 2-torus and lay it out flat, we get a rectangular sheet where opposite edges are identified, suggesting that perhaps we “ought” to be using a city-block metric or 1-norm aligned with the torus rotation directions, rather than the 2-norm $|\mathbf{B}|$. With the torus interpretation, some directions *are* special, so the use of the 2-norm based on an inner product seems somewhat questionable.

One way to overcome this might be to use a different method to find each torus to search over. Consider for a moment the usual Newton algorithm in normal Euclidean space. In the Newton approach we assume that the cost function $J(\mathbf{x})$ at a point \mathbf{x} can be approximated by a Taylor expansion about some minimum point \mathbf{x}^* , as $J(\mathbf{x}) = (\mathbf{x} - \mathbf{x}^*)^T \Sigma (\mathbf{x} - \mathbf{x}^*)$ where Σ is a symmetric Hessian matrix. Writing the eigenvector-eigenvalue decomposition $\Sigma = \mathbf{V} \Lambda \mathbf{V}^T$ where Λ is diagonal, we can write $J(\mathbf{x}) = \sum_i \lambda_i |\mathbf{v}_i^T (\mathbf{x} - \mathbf{x}^*)|^2 = \sum_i \lambda_i |y_i - y_i^*|^2$ where \mathbf{v}_i is the i th column of the eigenvector matrix \mathbf{V} , and $y_i = \mathbf{v}_i^T \mathbf{x}$. In one parallel update step $y_i \leftarrow y_i - \frac{dJ/dy_i}{d^2J/dy_i^2}$ for each y_i we would reach the bottom of J . Therefore, given the Hessian Σ , the eigenvector-eigenvalue decomposition allows us to jump directly to the minimum of J , by searching over each direction y_i independently.

We conjecture that it should be possible to choose the torus \mathbf{t} in an analogous way. In our torus, we might expect J not to have a simple Taylor expansion, but instead to have a multidimensional Fourier expansion, perhaps $J(\mathbf{B}) = J(t_1, \dots, t_m) = \sum_i \alpha_i (1 - \cos(t_i - t_i^*))$ where $\mathbf{B} = t_1 \mathbf{H}_1 + \dots + t_m \mathbf{H}_m$. Note that we might have to consider whether J repeats more often than $t_i = 2\pi$. As we mentioned earlier, the solutions to the standard ICA problem do repeat every $t = \pi$ since negative solutions are just as valid as positive solutions. For a lack of any better method, we could use the “steepest-descent” direction \mathbf{H} to generate this torus, but it would be interesting to see if it were possible to search for a torus which would directly contain the lowest value of J . As yet it is not clear how this could be evaluated, but we believe that the search for a method to identify this “best torus” is an interesting challenge.

11 Relation to other work

We have concentrated almost exclusively in this article on the problem of nonnegative ICA using orthogonality constraints, and have tried to avoid relying on much existing mathematical background. Other than the references we have already mentioned, there is an increasing amount of work in related areas and we will briefly touch on some of it here for the interested reader.

Many of the concepts discussed here can be generalized to the *Stiefel* and *Grassman* manifolds [12,35,26]. The group $SO(n)$ can be considered to be a special case of the Stiefel manifold: the Stiefel manifold is the space of rectangular $n \times p$ orthogonal matrices, while the Grassman manifold is concerned only with the subspace spanned by such matrices. See in particular the article by Edelman et al [25], which explains their conjugate gradient and Newton methods with examples in Matlab, and Fiori [6] for an introduction to these approaches for ICA-type problems.

The Lie group method has been extensively studied recently by e.g. Iserles, Munthe-Kaas and colleagues [17] for integration of differential equations over Lie groups (see [36] for an introductory article). Attempting to accurately integrate differential equations in this way can get into very heavy mathematics, which we are able to avoid in the present article precisely because we only wish to find a minimum of the function J constrained on a manifold, rather than exactly follow a particular differential equation. Any differential equation we might construct in order to minimize J is only “to point us in the right direction”, so to speak.

In this article we have worked in the Lie group $SO(n)$ and the Lie algebra

$\mathfrak{so}(n)$, using the matrix exponential to map from $\mathfrak{so}(n)$ to $SO(n)$. An alternative method is to use the Cayley transform: see e.g. [32,33] for details of this approach. Even though we have not considered it in detail here, it has been argued that avoiding the exponential itself makes the Cayley transform particularly useful for tasks on $SO(n)$ [17].

We should also mention that there are also applications in computer vision [37] and robotics [32] due to the constrained nature of physical systems; and earlier work by Fiori in the ICA direction draws an analogy between a neural system with constraints and a rigid mechanical system [38,39].

12 Conclusions

In this article we have explored the task of pre-whitened non-negative independent component analysis (nonnegative ICA) using orthogonality constraints. We discussed the concept of the manifold of square orthogonal matrices \mathbf{W} where $\mathbf{W}^T\mathbf{W} = \mathbf{W}\mathbf{W}^T = \mathbf{I}_n$, which form the special orthogonal Lie group $SO(n)$. This group has a corresponding Lie algebra $\mathfrak{so}(n)$, embodying of the derivatives of elements of $SO(n)$, with the exponential map from $\mathfrak{so}(n)$ to $SO(n)$.

We have seen that many of the assumptions that we make in our usual Euclidean space have to be revised when we work on manifolds in this way. Concepts such as length, gradient, straight line, parallel, and so on have to be reviewed in the light of the task of minimization of a cost function, and modified to suit the constraint. Other things that we take for granted in Euclidean space, such as the commutivity of movements in the space, also no longer hold

on the constraint manifold. Nevertheless it is still possible to adapt optimization methods such as steepest-descent search, and the well-known conjugate gradients algorithm, to work on this constraint surface. We can also develop new methods, such as the Fourier expansion equivalent of the Newton step, which work specifically in a compact space such as $\text{SO}(n)$.

By finding a toral subalgebra for $\mathfrak{so}(n)$, an Abelian (commutative) algebra formed from a subset of the points in $\mathfrak{so}(n)$, we can recover some of the properties such as commutativity that we are familiar with in Euclidean space. This also suggests a more efficient calculation of successive matrix exponentials, since the required matrix decomposition does not need to be performed each time.

In this article, I have deliberately avoided relying on too much prior mathematical background, concentrating instead on the intuitive concepts of the constraints and geometry involved. In this way I hope that researchers in ICA without an existing background in Lie groups or differential manifolds will nevertheless be able to see that the concepts are not as difficult as the notation sometimes appears, and will be interested enough to delve more deeply into these approaches.

Acknowledgements

I would like to thank three anonymous reviewers for valuable comments which helped me to improve this article.

This work was partially supported by Grant GR/R54620 from the UK Engineering and Physical Sciences Research Council.

References

- [1] M. D. Plumbley, Conditions for nonnegative independent component analysis, *IEEE Signal Processing Letters* 9 (6) (2002) 177–180.
- [2] A. Hyvärinen, J. Karhunen, E. Oja, *Independent Component Analysis*, John Wiley & Sons, 2001.
- [3] S. Amari, Natural gradient works efficiently in learning, *Neural Computation* 10 (2) (1998) 251–276.
- [4] S. I. Amari, A. Cichocki, Adaptive blind signal processing - Neural network approaches, *Proceedings of the IEEE* 86 (10) (1998) 2026–2048.
- [5] J.-F. Cardoso, B. H. Laheld, Equivariant adaptive source separation, *IEEE Transactions on Signal Processing* 44 (1996) 3017–3030.
- [6] S. Fiori, A theory for learning by weight flow on Stiefel-Grassman manifold, *Neural Computation* 13 (2001) 1625–1647.
- [7] G. H. Golub, C. F. van Loan, *Matrix Computations*, North Oxford Academic, Oxford, England, 1983.
- [8] E. Oja, A simplified neuron model as a principal component analyzer, *Journal of Mathematical Biology* 15 (1982) 267–273.
- [9] R. W. Brockett, Dynamical systems that sort lists, diagonalize matrices, and solve linear programming problems, *Linear Algebra and its Applications* 146 (1991) 79–91.
- [10] L. Xu, Least mean square error reconstruction principle for self-organizing neural-nets, *Neural Networks* 6 (5) (1993) 627–648.
- [11] E. Oja, M. D. Plumbley, Blind separation of positive sources by globally convergent gradient search, *Neural Computation* 16 (9) (2004) 1811–1825.

- [12] S. C. Douglas, S.-Y. Kung, An ordered-rotation Kuicnet algorithm for separating arbitrarily-distributed sources, in: Proceedings of the International Workshop on Independent Component Analysis and Blind Signal Separation (ICA'99), Aussois, France, 1999, pp. 81–86.
- [13] S. C. Douglas, Self-stabilized gradient algorithms for blind source separation with orthogonality constraints, *IEEE Transactions on Neural Networks* 11 (6) (2000) 1490–1497.
- [14] J. G. Belinfante, B. Kolman, H. A. Smith, An introduction to Lie groups and Lie algebras, with applications, *SIAM Review* 8 (1) (1966) 11–46.
- [15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd Edition, Cambridge University Press, 1992.
- [16] M. D. Plumbley, Optimization using Fourier expansion over a geodesic for non-negative ICA, in: Proceedings of the International Conference on Independent Component Analysis and Blind Signal Separation (ICA 2004), Granada, Spain, 2004, (To appear).
- [17] A. Iserles, H. Z. Munthe-Kaas, S. P. Nørsett, A. Zanna, Lie-group methods, *Acta Numerica* 9 (2000) 215–365.
- [18] B. Schutz, *Geometrical Methods of Mathematical Physics*, Cambridge University Press, Cambridge, UK, 1980.
- [19] Y. Nishimori, Learning algorithm for ICA by geodesic flows on orthogonal group, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN'99), Vol. 2, Washington, DC, 1999, pp. 933–938.
- [20] C. Moler, C. V. Loan, Nineteen dubious ways to compute the exponential of a matrix, *SIAM Review* 20 (4) (1978) 801–836.

- [21] M. D. Plumbley, Algorithms for nonnegative independent component analysis, *IEEE Transactions on Neural Networks* 14 (3) (2003) 534–543.
- [22] E. M. E. Wermuth, Two remarks on matrix exponentials, *Linear Algebra and its Applications* 117 (1989) 127–132.
- [23] S. Fiori, R. Rossi, Stiefel-manifold learning by improved rigid-body theory applied to ICA, *International Journal of Neural Systems* 13 (5) (2003) 273–290.
- [24] J. Gallier, D. Xu, Computing exponentials of skew-symmetric matrices and logarithms of orthogonal matrices, *International Journal of Robotics and Automation* 18 (1) (2003) 10–20.
- [25] A. Edelman, T. A. Arias, S. T. Smith, The geometry of algorithms with orthogonality constraints, *SIAM J. Matrix Anal. Appl.* 20 (2) (1998) 303–353.
- [26] R. Martin-Clemente, C. G. Puntonet, J. I. Acha, Blind signal separation based on the derivatives of the output cumulants and a conjugate gradient algorithm, in: T.-W. Lee, T.-P. Jung, S. Makeig, T. J. Sejnowski (Eds.), *Proceedings of the International Conference on Independent Component Analysis and Signal Separation (ICA2001)*, San Diego, California, 2001, pp. 390–393.
- [27] J. R. Shewchuk, An introduction to the conjugate gradient method without the agonizing pain, unpublished draft. Available at URL <http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.ps> (Aug. 1994).
- [28] E. Celledoni, A. Iserles, Methods for the approximation of the matrix exponential in a Lie-algebraic setting, *IMA J. Num. Anal.* 21 (2001) 463–488, also available as DAMPT Numerical Analysis Report NA1999/03, Department of Applied Mathematics and Theoretical Physics, University of Cambridge.

- [29] A. Iserles, A. Zanna, Efficient computation of the matrix exponential by generalized polar decompositions, Numerical Analysis Report NA2002/09, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, to appear in SIAM J Num. Anal. (2002).
- [30] R. B. Taher, M. Rachidi, Some explicit formulas for the polynomial decomposition of the matrix exponential and applications, *Linear Algebra and its Applications* 350 (1-3) (2002) 171–184.
- [31] L. Lopez, C. Mastroserio, T. Politi, Newton-type methods for solving nonlinear equations on quadratic matrix groups, *Journal of Computational and Applied Mathematics* 115 (1-2) (2000) 357–368.
- [32] J. H. Manton, Optimization algorithms exploiting unitary constraints, *IEEE Transactions on Signal Processing* 50 (3) (2002) 635–650.
- [33] I. Yamada, T. Ezaki, An orthogonal matrix optimization by dual Cayley parametrization technique, in: Proc. 4th Intl. Symp. On Independent Component Analysis and Blind Signal Separation (ICA2003), Nara, Japan, 2003, pp. 35–40.
- [34] S. Fiori, Fixed-point neural independent component analysis algorithms on the orthogonal group, *Journal of Future Generation Computer Systems* Accepted for publication.
- [35] K. Rahbar, J. P. Reilly, Blind source separation algorithm for MIMO convolutive mixtures, in: T.-W. Lee, T.-P. Jung, S. Makeig, T. J. Sejnowski (Eds.), *Proceedings of the International Conference on Independent Component Analysis and Signal Separation (ICA2001)*, San Diego, California, 2001, pp. 242–247.
- [36] A. Iserles, Brief introduction to Lie-group methods, in: D. Estep, S. Tavener (Eds.), *Collected Lectures on the Preservation of Stability Under Discretization*

(Proceedings in Applied Mathematics Series), SIAM, Philadelphia, 2002, Ch. 7, pp. 123–143.

[37] Y. Ma, A differential geometric approach to multiple view geometry in spaces of constant curvature, *International Journal of Computer Vision* 58 (1) (2004) 37–53.

[38] S. Fiori, ‘Mechanical’ neural learning for blind source separation, *Electronics Letters* 35 (22) (1999) 1963–1964.

[39] S. Fiori, A theory for learning based on rigid bodies dynamics, *IEEE Transactions on Neural Networks* 13 (3) (2002) 521–531.