# Parallel implementations of Riemannian conjugate gradient methods for joint approximate diagonalization*

Nela Bosner[†]

*Department of Mathematics, Faculty of Science, University of Zagreb, Bijenička cesta 30, HR-10 000 Zagreb, Croatia*

Received November 25, 2024; accepted December 21, 2024

**Abstract.** We propose multi-threaded parallel implementations of the Riemannian conjugate gradient method (CG) on the Stiefel manifold and on the oblique manifold, suitable for solving two forms of the joint approximate diagonalization problem. In our implementations, each fundamental step of the method is explicitly modified and parallelized to enhance computational efficiency. Numerical experiments demonstrate that our modified CG implementations are more efficient than the original versions.

**AMS subject classifications**: 15A21, 15A23, 58C05, 58C25, 47J25, 65J15, 65K10, 65Y05, 65Y20

**Keywords**: joint approximate diagonalization, Riemannian conjugate gradient method, matrix manifolds, parallel implementations, efficient algorithms

## 1. Introduction

We are solving the following problem: Given a set of symmetric matrices $A^{(1)}$, $A^{(2)}$, $\dots$, $A^{(m)} \in \mathbb{R}^{n\times n}$, find orthogonal or nonsingular $Y \in \mathbb{R}^{n\times n}$ such that

$$A^{(p)} = Y \cdot D^{(p)} \cdot Y^T, \quad \text{for all } p = 1, \dots, m, \tag{1}$$

where $D^{(p)}$ are either diagonal or as diagonal as possible when exact diagonalization is not feasible. The latter case is referred to as "joint approximate diagonalization". In practice, there are two variants of diagonalization: the solution $Y$ must be orthogonal and the problem is equivalent to joint eigenvalue decompositions of the input matrices, or $Y$ is only nonsingular. Each of these variants requires a different approach.

Joint approximate diagonalization (JAD) of multiple matrices in form (1) is a core problem in numerous applications, including canonical polyadic decomposition of a symmetric tensor which has a symmetric orthogonal factorization [21, 22], blind source separation [15, 25]; blind beam forming [13], estimation of frequencies of exponentials in noise [24].

Several approaches are commonly used in practice to solve this problem. Basically, the problem of joint diagonalization reduces to finding $X = Y^{-T}$ such that

$$X^T A^{(p)} X = D^{(p)}, \quad p = 1, \dots, m.$$

*This work was supported by the Croatian Science Foundation under grant no. IP-2019-04-6268
[†]Corresponding author. *Email address:* nela.bosner@math.hr (N. Bosner)

To compute the orthogonal joint diagonalizing matrices, Jacobi-type methods were proposed in [12, 13, 14]. In [18], eigenvalue decomposition of a random linear combination of matrices $A^{(p)}$ is used to obtain joint diagonalization of symmetric matrices.

The problem can also be treated as Riemannian minimization of an appropriately chosen function that measures the diagonality of $X^T A^{(p)} X$ on a special matrix manifold, as described in [16]. In this case, it is the Stiefel manifold (the orthogonal group), and the manifold represents a constraint to the optimization problem, where Riemannian minimization methods are adapted to utilize manifold geometry. By exploiting differential–geometric tools, classical Euclidean optimization methods, such as steepest descent, conjugate gradient, BFGS and Newton's methods, are generalized to this non-Euclidean domain. The most exploited objective function is

$$F(X) = \frac{1}{2}\sum_{p=1}^{m} \|\mathrm{Off}(X^T A^{(p)} X)\|_F^2, \tag{2}$$

where for a matrix $B = [b_{ij}] \in \mathbb{R}^{n\times n}$, $\mathrm{Off}(B) = B - \mathrm{Diag}(B)$, and $\mathrm{Diag}(B) = \mathrm{diag}(b_{11}, \ldots, b_{mm})$. Such approach, with the conjugate gradient and the Newton's method, is proposed in [1, 28], where the complexity of the described method can be quite high. A general Riemannian BFGS algorithm applied to the Stiefel manifold and another optimization problem is described in [27]. Both Newton's and BFGS methods require the solution of a large linear system or a specific matrix linear equation at each iteration.

On the other hand, there are other non-Riemannian approaches, such as the methods described in [23, 35]. These algorithms are derived by reformulating the constrained optimization problem as an unconstrained one, written in terms of a local parametrization at each iteration. The problem with such algorithms is that they do not fully exploit the special properties arising from the structure of the manifold. Additionally, they suffer from slow convergence and high computational complexity, as they require solving linear systems and computing singular value decompositions in each iteration. In some cases, these algorithms also become numerically unstable.

In the case of non-orthogonal diagonalization, it is required that $X$ be nonsingular. This problem is challenging since it does not rely on eigenvalue decomposition. The transformation in (1) coincides with congruence, not similarity. Jacobi type methods also exist for this case, as proposed in [5, 32].

The other class of methods minimizes objective functions that measure the diagonality of $X^T A^{(p)} X$ under various constraints, which require certain heuristic corrections to prevent solution approximation to converge toward a zero-matrix. This approach typically reduces the original optimization problem to a sequence of subproblems, as described in [34, 36, 37].

There is also a possibility of using different measures for the diagonality of matrices $X^T A^{(p)} X$ implemented in the objective function, as proposed in [17, 26, 7, 6]. In special cases, these functions allow for efficient estimation, but in most cases, they are computationally expensive.

Finally, there is a Riemannian minimization approach for determining a non-orthogonal diagonalizing matrix, where the problem of finding an appropriate manifold with a simple structure is quite challenging. In [3], the authors propose the oblique manifold (a set of matrices with unit column norms) for solving blind source separation or independent component analysis problems. The Riemannian structure of that manifold is simple, and its differential-geometric tools are computationally inexpensive. The objective function in [3] is also of form (2), and the trust-region method is chosen as the method for Riemannian minimization. This method solves a trust-region minimization subproblem in each iteration, where the size of the subproblem is equal to the dimension of the manifold, see [2]. Newton's method on the oblique manifold is described in [20]. As previously mentioned, it requires solving a large linear system at each iteration. On the other hand, the conjugate gradient method requires the solution of a one-dimensional problem in each iteration, referred to as line search. Its application to the oblique manifold is described in [31], though with a different objective function not related to joint diagonalization. Besides the oblique manifold, some authors have studied Riemannian minimization on other matrix manifolds with more complex structures, see e.g. [9, 11, 10].

Most of the methods listed above have two fundamental problems from a numerical point of view: slow convergence, especially for a large number of matrices, and large computational complexity. In this paper, we address the second problem by selecting matrix manifolds and objective functions with favorable numerical properties, and by combining their characteristics in order to produce highly efficient algorithms. Our approach relies on Riemannian optimization on matrix manifolds, which offers an opportunity for efficient parallel implementation, particularly for larger dimension $n$ and a large number of input matrices $m$. We consider the Riemannian structure of the manifold, along with the evaluation of the objective function, its gradient, and its Hessian. We also want to control accuracy of line search in our algorithms. Further, we aim to reduce or even avoid computing matrix inverses, solving linear systems, and performing matrix factorizations such as eigenvalue and singular value decompositions in each iteration. Therefore, we chose the **conjugate gradient (CG) method** to minimize function (2) on two matrix manifolds: the **Stiefel manifold** and the **oblique manifold**.

The conjugate gradient algorithm for minimization on Riemannian manifolds is nicely described in [33]. As noted in [4] and [16], "conjugate gradient techniques are considered because they are easy to implement, have low storage requirements" and lower numerical cost, and provide stronger global convergence properties than the Newton method while achieving superlinear local convergence, as proved in [33]. For the particular choice of objective function and manifolds in this paper, the conjugate gradient method is very simple for implementation, exploits simple matrix operations and is suitable for parallelization. Furthermore, it has "cheaper" iterations compared to other commonly used optimization methods, such as BFGS, Newton's and the trust-region method, since it does not require solving linear systems. The choice of the objective function, the manifolds, and the optimization method was crucial for developing efficient algorithms. In particular, we considered variants of the method that are often avoided due to their numerical complexity, but may exhibit superior convergence properties. Therefore, we have implemented many variants of the method with different complexities and convergence properties, and we provide a recommendation for the most efficient one.

The paper is organized as follows. In Section 2, we give a brief overview of basic concepts from differential geometry required by the conjugate gradient method. This section is divided into two subsections: one focuses on the specific objective function and its derivatives, while the other covers the geometric structures of the Stiefel and oblique manifolds. Section 3 presents our multi-threaded parallel implementations of the conjugate gradient algorithm on these two matrix manifolds, where we developed and implemented many variants of the algorithm. The same section also describes procedures of operation reorganization and explicit parallelization. The computational complexity of the specific variants is analyzed in Section 4. Section 5 presents the results of numerical experiments, which confirm the superiority of the parallel versions. Additionally, the convergence is analyzed, and a recommendation for the most efficient variant is proposed. Finally, Section 6 concludes the paper.

## 2. Conjugate gradient method on Riemannian manifolds

Riemannian optimization algorithms are adapted to exploit the geometry of the manifold. Therefore, standard concepts and operations are tailored to the manifold structure to ensure that approximations in all iterations remain on the manifold. The concepts required by the algorithms are as follows: the definition of the inner product in tangent spaces called the Riemannian metric, line search, where straight lines are replaced by curves called geodesics or retractions; parallel translation, where a tangent vector is transported from one point on the manifold to another, the gradient, which is a tangent vector, and the Hessian, which acts on tangent vectors. These basic concepts from differential geometry related to manifolds can be found in [4], and they represent the building blocks of the conjugate gradient method on Riemannian manifolds.

Since we deal with matrix manifolds and matrices are typically denoted by capital letters, we will adopt this convention in our notation. As described in [4, 16, 29, 33], let $\mathcal{M}$ be a matrix manifold and $F : \mathcal{M} \to \mathbb{R}$ an objective function; the conjugate gradient method computes an approximation to the solution of the optimization problem $\min_{X\in\mathcal{M}} F(X)$. It is an iterative method based on line search, where the search direction in each iteration is specified as the steepest descent direction, orthogonalized against the previous search direction in a certain scalar product determined by the Hessian and the Riemannian metric (the property of conjugacy).

In the $k$-th iteration, the method produces the current solution approximation $X_k$ and search direction $H_k$ required to reach the next approximation. In the idealized version of the method, these two data determine a new geodesic $\Gamma_k$ going through $\Gamma_k(0) = X_k$ in direction $\dot{\Gamma}_k(0) = H_k$, and the new approximation $X_{k+1}$ is computed as $X_{k+1} = \Gamma_k(t_{k+1})$, where $t_{k+1}$ is a local minimum of $F(\Gamma_k(t))$ (the line search step). It only remains to compute a new search direction $H_{k+1}$, and this is done in the following way: compute gradient $G_{k+1} = \text{grad } F(X_{k+1})$, and choose

$$H_{k+1} = -G_{k+1} + \beta_{k+1}\Pi(H_k),$$

where $\Pi(H_k) = P_{\Gamma_k}^{t_{k+1}\leftarrow 0} H_k$ is the parallel translation of $H_k$ along $\Gamma_k$ to $X_{k+1}$. Parameter $\beta_{k+1}$ is chosen such that $H_{k+1}$ and $\Pi(H_k)$ are conjugated; namely,

$$\langle \text{Hess } F(X_{k+1})[H_{k+1}], \Pi(H_k)\rangle_{X_{k+1}} = 0,$$

where $\langle\ ,\ \rangle_X$ is the Riemannian metric in the tangent space at the point $X$ on the manifold. From the property of exact conjugacy it follows that

$$\beta_{k+1} = \frac{\langle \text{Hess } F(X_{k+1})[G_{k+1}], \Pi(H_k)\rangle_{X_{k+1}}}{\langle \text{Hess } F(X_{k+1})[\Pi(H_k)], \Pi(H_k)\rangle_{X_{k+1}}}.$$

Choosing $H_{k+1} = -G_{k+1}$ corresponds to the steepest descent method, whose convergence is proven in [33] to be only linear. In contrast, the same paper demonstrates that the conjugate gradient method achieves superlinear convergence. Additionally, in [30], numerical experiments on various matrix manifolds confirm that the steepest descent method is the slowest. Consequently, this choice of the search direction generally accelerates the slow convergence of the steepest descent method.

This idealized version of the conjugate gradient method is usually relaxed in order to improve computational efficiency. In many applications, certain operations in the described algorithm are replaced with simpler variants. The following list outlines typical alternative approaches:

- Since computing a new point on the geodesic is numerically expensive for some manifolds, retractions are used instead. Similarly, simpler vector transport is used as a substitute for parallel translation.
- As stated in [4], if the numerical cost of computing the exact line search solution is not prohibitive, then the minimizing value $t_{k+1}$ should be used. Otherwise, an approximation is used instead.
- The condition of exact conjugacy is replaced by some approximation that avoids computing the Hessian. As described in [16, 33], typical choices for parameter $\beta_{k+1}$ are derived from the finite difference approximation to the Hessian, exploiting the assumption that $X_{k+1}$ is a minimum point along the geodesic, and they include

$$\beta_{k+1} = \frac{\langle G_{k+1} - \Pi(G_k), G_{k+1}\rangle}{\langle G_k, G_k\rangle}, \qquad \text{the Polak–Riebière formula,} \tag{3}$$

$$\beta_{k+1} = \frac{\langle G_{k+1}, G_{k+1}\rangle}{\langle G_k, G_k\rangle}, \qquad \text{the Fletcher–Reeves formula.} \tag{4}$$

All these alternatives are chosen to ensure convergence, though they may slow down the convergence speed in some cases, as demonstrated by the results of our numerical experiments in Section 5. An interesting convergence analysis of several variants of the Riemannian conjugate gradient method is presented in [30], concerning non-exact line search and approximations of exact conjugacy.

On the other hand, for our choice of the objective function and the matrix manifolds, we are able to compute the exact Hessian and to obtain the exact conjugacy of the search direction in a less complex way.

## 2.1. Objective function

Let $A^{(p)} = [a_{ij}^{(p)}] \in \mathbb{R}^{n\times n}$, $p = 1, \ldots, m$ be a set of symmetric matrices, and let $X = [x_{ij}] \in \mathbb{R}^{n\times n}$ be a nonsingular or orthogonal matrix. Then we define our objective function as

$$F(X) = \frac{1}{2}\sum_{p=1}^{m} \| \operatorname{Off}(X^T A^{(p)} X)\|_F^2 = \frac{1}{2}\sum_{p=1}^{m} \sum_{\substack{i,j=1 \\ i\neq j}}^{n} \left( \sum_{k,\ell=1}^{n} x_{ki} a_{k\ell}^{(p)} x_{\ell j} \right)^2 . \tag{5}$$

The conjugate gradient algorithm requires knowledge of the gradient and, potentially, the Hessian at each approximation on the manifold. Therefore, the first and the second partial derivatives of the function $F$ must be computed in each iteration. We adopt simpler notations from [16], where for tangent vectors $\Xi_\ell = [\xi_{ij}^{(\ell)}]$, $\ell = 1, 2$,

$$F_X(X) := \nabla F(X) = \left[ \frac{\partial F}{\partial x_{rs}}(X) \right]_{rs},$$
$$F_{XX}(X)(\Xi_1, \Xi_2) := \sum_{r,s=1}^{n} \sum_{u,v=1}^{n} \frac{\partial^2 F}{\partial x_{uv} \partial x_{rs}}(X) \xi_{rs}^{(1)} \xi_{uv}^{(2)}.$$

It is easy to see that for the function defined in (5) we have

$$F_X(X) = 2\sum_{p=1}^{m} A^{(p)} X \cdot \operatorname{Off}(X^T A^{(p)} X), \tag{6}$$

$$\begin{aligned} F_{XX}(X)(\Xi_1, \Xi_2) = 2\sum_{p=1}^{m} \Big[ &\operatorname{tr}(\Xi_1^T A^{(p)} \Xi_2 \cdot \operatorname{Off}(X^T A^{(p)} X)) \\ &+ \operatorname{tr}(X^T A^{(p)} \Xi_1 \cdot \operatorname{Off}(X^T A^{(p)} \Xi_2)) + \operatorname{tr}(\Xi_1^T A^{(p)} X \cdot \operatorname{Off}(X^T A^{(p)} \Xi_2)) \Big], \end{aligned} \tag{7}$$

where for $B = [b_{ij}] \in \mathbb{R}^{n\times n}$, $\operatorname{tr}(B) = \sum_{i=1}^{n} b_{ii}$. See also [3, 20]. We emphasize here that $F_X(X)$ and $F_{XX}(X)$ are obtained as sums of either matrix products or traces of matrix products involving input matrices, which is suitable for parallelization, since all these products can be computed independently.

## 2.2. Matrix manifolds

In our problem, the diagonalizing matrix $X$ is square, so the Stiefel manifold is, in fact, the orthogonal group. A formal definition is

$$\mathcal{S}_{n,n} = \{X \in \mathbb{R}^{n\times n} : X^T X = I\}.$$

A formal definition of the oblique manifold is as follows:

$$\mathcal{O}_{n,n} = \{X \in \mathbb{R}^{n\times n} : \operatorname{Diag}(X^T X) = I\}.$$

The oblique manifold is chosen for its simple structure, and in the case of non-orthogonal joint diagonalization, it provides a solid constraint that prevents solution approximation from converging to the zero matrix. Since joint diagonalization is invariant under column scaling of the diagonalizing matrix $X$, selecting a solution from the oblique manifold fixes this scaling.

Next, we list the required concepts from differential geometry regarding both manifolds; for details, see [4, 16, 20].

- Dimension: $\dim(\mathcal{S}_{n,n}) = n^2 - \frac{n(n+1)}{2} = \frac{n(n-1)}{2}$, $\dim(\mathcal{O}_{n,n}) = n^2 - n = n(n-1)$.
- Tangent vector $\Xi \in T_X\mathcal{M}$: $\begin{array}{l} \mathcal{M} = \mathcal{S}_{n,n} : \Xi = X \cdot A \text{ with } A^T = -A, \\ \mathcal{M} = \mathcal{O}_{n,n} : \text{satisfies } \mathrm{Diag}(X^T\Xi) = 0. \end{array}$
- Riemannian metric: $\langle \Xi, \Theta \rangle_X = \mathrm{tr}(\Xi^T\Theta) = \mathrm{vec}(\Xi)^T\mathrm{vec}(\Theta), \quad \Xi, \Theta \in T_X\mathcal{M}$.
- Geodesic $\Gamma$ such that $\Gamma(0) = X$ and $\dot{\Gamma}(0) = H$:

$$\mathcal{M} = \mathcal{S}_{n,n} : \Gamma(t) = Xe^{At}, \ H = XA, \ A^T = -A, \tag{8}$$

$$\begin{aligned} \mathcal{M} = \mathcal{O}_{n,n} : \Gamma(t) &= X\cos(\Lambda t) + H\Lambda^{-1}\sin(\Lambda t), \\ \Lambda &= \mathrm{diag}\ (\|H(:,1)\|_2, \ldots, \|H(:,n)\|_2). \end{aligned} \tag{9}$$

- Parallel vector field $\Xi(t)$ such that $\Xi(0) = \Xi_0 \in T_X\mathcal{M}$ along geodesic $\Gamma$:

$$\mathcal{M} = \mathcal{S}_{n,n} : \Xi(t) = Xe^{\frac{At}{2}}Be^{\frac{At}{2}}, \ \Xi_0 = XB, \ B^T = -B, \tag{10}$$

$$\begin{aligned} \mathcal{M} = \mathcal{O}_{n,n} : \Xi(t) &= \Xi_0 - X\Lambda^{-1}M\sin(\Lambda t) + H\Lambda^{-2}M\cos(\Lambda t) - H\Lambda^{-2}M, \\ M &= \mathrm{diag}\ (H(:,1)^T\Xi_0(:,1), \ldots, H(:,n)^T\Xi_0(:,n)). \end{aligned} \tag{11}$$

- Retraction $\Gamma$ such that $\Gamma(0) = X$ and $\dot{\Gamma}(0) = H$:

$$\mathcal{M} = \mathcal{S}_{n,n} : \text{Let } (X + tH) = Q_tR_t \text{ be QR factorization, then } \Gamma(t) = Q_t, \tag{12}$$

$$\begin{aligned} \mathcal{M} = \mathcal{O}_{n,n} : \Gamma(t) &= (X + tH)N(t)^{-1}, \\ N(t) &= \mathrm{diag}\ (\|(X + tH)(:,1)\|_2, \ldots, \|(X + tH)(:,n)\|_2)\ . \end{aligned} \tag{13}$$

- Vector transport $\Xi(t)$ such that $\Xi(0) = \Xi_0 \in T_X\mathcal{M}$ along retraction $\Gamma$:

$$\mathcal{M} = \mathcal{S}_{n,n} : \Xi(t) = \tfrac{1}{2}Q_t(Q_t^T\Xi_0 - \Xi_0^TQ_t), \tag{14}$$

$$\begin{aligned} \mathcal{M} = \mathcal{O}_{n,n} : \Xi(t) &= \Xi_0 - (X + tH)N(t)^{-2}M(t) = \Xi_0 - \Gamma(t)N(t)^{-1}M(t), \\ M(t) &= \mathrm{diag}\ ((X + tH)(:,1)^T\Xi_0(:,1), \ldots, (X + tH)(:,n)^T\Xi_0(:,n)). \end{aligned} \tag{15}$$

- Gradient of $F : \mathcal{M} \to \mathbb{R}$:

$$\mathcal{M} = \mathcal{S}_{n,n} : \mathrm{grad}\ F(X) = \frac{1}{2}(F_X(X) - XF_X(X)^TX), \tag{16}$$

$$\mathcal{M} = \mathcal{O}_{n,n} : \mathrm{grad}\ F(X) = F_X - X\mathrm{Diag}(X^TF_X(X)). \tag{17}$$

- Riemannian Hessian of $F : \mathcal{M} \to \mathbb{R}$:

$$\begin{aligned} \mathcal{M} = \mathcal{S}_{n,n} : \langle \mathrm{Hess}\ F(X)[\Xi_1], \Xi_2 \rangle_X &= F_{XX}(\Xi_1, \Xi_2) - \\ &- \tfrac{1}{2}\,\mathrm{tr}((F_X(X)^TX + X^TF_X(X))\Xi_1^T\Xi_2), \end{aligned} \tag{18}$$

$$\begin{aligned} \mathcal{M} = \mathcal{O}_{n,n} : \langle \mathrm{Hess}\ F(X)[\Xi_1], \Xi_2 \rangle_X &= F_{XX}(\Xi_1, \Xi_2) - \\ &- \mathrm{tr}(F_X(X)^TX\mathrm{Diag}(\Xi_1^T\Xi_2)). \end{aligned} \tag{19}$$

## 3. Efficient implementations of conjugate gradient methods on two matrix manifolds

The most expensive operation in the conjugate gradient method on the Stiefel and oblique manifolds is the computation of the matrix exponential required for evaluation of the geodesic on $\mathcal{S}_{n,n}$. Of course, a simpler retraction can be used instead of the geodesic. In our numerical experiments, we will assess the convergence of the algorithms based on these two curves in terms of the number of iterations and execution time. Our goal is to exploit the beneficial properties of the geodesic while reducing the cost of its computation. All other tasks reduce to basic matrix operations: matrix multiplication and sum, the trace of a matrix, and extracting diagonal elements of a matrix. Additionally, we observe that the same products are involved in computations of different terms. Therefore, we can reorganize our algorithm to compute these products only once, store them, and reuse them. We begin with an efficient implementation of the objective function evaluation and its derivatives.

The objective function evaluation is indirectly required in line search, and more commonly in stopping criteria. It is important to note that matrix products $X^T A^{(p)} X$ for all $p$ can be computed in parallel.

When computing the standard gradient $F_X(X)$ in form (6), the products $A_X^{(p)} = A^{(p)} \cdot X$ and $B^{(p)} = X^T \cdot A_X^{(p)}$ for all $p = 1, \ldots, m$ are computed in parallel, and the results are stored for later use. The final products $A_X^{(p)} \cdot \mathrm{Off}(B^{(p)})$ are also computed in parallel.

It is worth noting that in both function and standard gradient evaluations, batched matrix multiplication functions can be utilized. In our implementation, we explicitly distribute the computation of multiple matrix-matrix products across the threads in the function `multiple_gemms()`. Threads use the BLAS ([8]) function `dgemm()` for matrix-matrix multiplication. Our function takes a set of matrices $A^{(p)}$ and a matrix $X$, or optionally another set of matrices $B^{(p)}$, and computes $C^{(p)} = scal \cdot A^{(p)} \cdot X$, or $C^{(p)} = scal \cdot X \cdot A^{(p)}$, or $C^{(p)} = scal \cdot A^{(p)} \cdot B^{(p)}$, for $p = 1, \ldots, m$.

Since computing the gradient is one of the first operations in each conjugate gradient iteration, and since $\mathrm{Off}(B^{(p)})$ is involved in both computations of $F_X(X)$ and $F_{XX}(X)$, as soon as these products are computed their diagonals are removed, stored and returned later when needed. At the end, the products are summed up, again, in a parallel manner, where $n^2$ elements of the result are uniformly distributed across the threads. When computing the second derivative function $F_{XX}(\Xi_1, \Xi_2)$ in form (7), the products $\Delta_1^{(p)} = (A_X^{(p)})^T \cdot \Xi_1$, $\Delta_2^{(p)} = (A_X^{(p)})^T \cdot \Xi_2$, and $\Delta_3^{(p)} = \Xi_1^T A^{(p)} \Xi_2$ for all $p$ are computed in parallel by function `multiple_gemms()`.

Furthermore, the traces of the products $\mathrm{tr}(\Delta_3^{(p)} \cdot \mathrm{Off}(B^{(p)}))$, $\mathrm{tr}(\Delta_1^{(p)} \cdot \mathrm{Off}(\Delta_2^{(p)}))$, and $\mathrm{tr}((\Delta_1^{(p)})^T \cdot \mathrm{Off}(\Delta_2^{(p)}))$ are computed in parallel, without first computing the matrix products and then performing trace evaluation. In our implementation, only diagonal elements of the products are efficiently evaluated and summed by function `traces_by_ddots()`. The same approach is used for the evaluation of $F(X)$. This function takes two sets of matrices $A^{(p)}$ and $B^{(p)}$, $p = 1, \ldots, m$, and computes $\sum_{p=1}^m \mathrm{tr}((A^{(p)})^T \cdot B^{(p)})$ or $\sum_{p=1}^m \mathrm{tr}(A^{(p)} \cdot B^{(p)})$.

Since the matrices are stored in one-dimensional arrays in column-major representation, the `traces_by_ddots()` function computes traces as one call to the `ddot()` BLAS function for the first sum, and with $n$ calls to `ddot()` for the second sum. The set $\{1, \ldots, k\}$, where $k = m$ or $k = mn$ is uniformly partitioned across the threads, and each thread sums a local partial sum of traces. At the end of the parallel evaluation of the partial sums, a single thread sums up all partial sums into the final sum.

## 3.1. Implementation of the method on the Stiefel manifold

The four terms required by the conjugate gradient algorithm, that are typical of the specific manifold, are the gradient, the Riemannian Hessian, the geodesic or retraction, and parallel translation along geodesic or vector transport along retraction. The gradient (16) and the Riemannian Hessian (18) are easily obtained by matrix multiplications and trace evaluation of the matrix product, which is computed by the function `traces_by_ddots()`.

A more complex task is the evaluation of the geodesic. The geodesic is involved in the one-dimensional optimization problem within line search

$$\min_{t\geq 0} F(\Gamma_k(t)), \tag{20}$$

where $\Gamma_k$ is the geodesic in the $k$-th iteration of the algorithm, starting at the $k$-th solution approximation $\Gamma_k(0) = X_k$ in the direction $\dot{\Gamma}_k(0) = H_k$. The direction vector is a tangent vector, which takes the form $H_k = X_k A_k$, where $A_k^T = -A_k$. From (8), we know the form of the geodesic, so $\Gamma_k(t) = X_k e^{A_k t}$. To solve this optimization problem, we will use an unconstrained optimization method that does not rely on derivatives. Prior to solving, we will search for an appropriate form for the composition of functions $F(\Gamma_k(t))$:

$$F_{\Gamma_k}(t) = F(\Gamma_k(t)) = \frac{1}{2}\sum_{p=1}^{m} \mathrm{tr}(\mathrm{Off}((e^{A_k t})^T B_k^{(p)} e^{A_k t})). \tag{21}$$

$B_k^{(p)} = X_k^T A^{(p)} X_k$ are stored previously in the $k$-th iteration, while computing $F_X(X_k)$. $e^{A_k t}$ are the only terms that depend on $t$, so only they are evaluated for different values of this parameter. Every optimization method for solving the line search problem (20) requires multiple evaluations of $e^{A_k t}$ for different parameters $t$, but for the same matrix $A_k$. Since this is the most expensive operation, we have adapted evaluation of the matrix exponential to meet the requirements of the one-dimensional optimization method. The matrix exponential evaluation is based on the scaling and squaring method described in [19] and implemented in MATLAB, which computes the Padé approximation of the function. In [19], it is shown that computation of the rational approximant for $e^Y$ requires only even powers of $Y$, and for double precision accuracy it is sufficient to compute only $Y_2 = Y^2$, $Y_4 = Y_2^2$ and $Y_6 = Y_2 Y_4$. In the case of geodesic, in our function `ss_alg_exp()` we only compute $A_{k,2} = A_k^2$, $A_{k,4} = A_{k,2}^2$ and $A_{k,6} = A_{k,2}A_{k,4}$ once and then we use it as $t^2 A_{k,2}$, $t^4 A_{k,4}$ and $t^6 A_{k,6}$ for different parameters $t$. The only drawback of this algorithm is the final evaluation of the rational approximant, which involves solving a linear system. Consequently, the frequency of matrix exponential evaluations is typically minimized. For each new parameter $t$, we perform additional matrix summations,up to 4 additional matrix multiplications, and solve a linear system. In our efficient implementation, the numerator and denominator polynomials are computed in parallel, with $n^2$ elements of the results uniformly distributed across the threads.

The function that implements the evaluation of $F(\Gamma_k(t))$, where $\Gamma_k$ is a geodesic as described above, is denoted by $F_\Gamma$ in pseudocode presented in Algorithm 1 and implemented as the function `F_geod()`.

Finally, we address the parallel translation of the search direction $H_k$. Since $H_k$ is a derivative of the geodesic $\Gamma_k$ at 0, the parallel vector field defining parallel translation is exactly $\dot{\Gamma}_k(t)$, so $\Pi(H_k) = \dot{\Gamma}_k(t_{k+1}) = X_k e^{A_k t_{k+1}} A_k = X_{k+1} \cdot A_k$, which requires only a single matrix multiplication.

Less numerically complex is line search along retraction. The function that implements the evaluation of $F(\Gamma_k(t))$, where $\Gamma_k$ is a retraction, is implemented by the function `F_retr()`. It employs the BLAS function for QR factorization.

**Algorithm 1** Parallel CG on the Stiefel manifold

**Require:** functions $F$, $F_X$, $F_{XX}$, $F_\Gamma(t) = F(\Gamma(t))$, $X_0 \in \mathcal{S}_{n,n}$, parameters *curve* indicating choice of line search curve, *linesearch* indicating line search method, *conj* indicating conjugacy formula.

```
1:  Compute $A^{(p)}_{X,0} = A^{(p)} X_0$, $B^{(p)}_0 = (A^{(p)}_{X,0})^T X_0$, for $p = 1, \dots, m$;
2:  $F_{X,0} = F_X(A^{(p)}_{X,0}, B^{(p)}_0)$; $F_0 = F(B^{(p)}_0)$; $G_0 = \frac{1}{2}(F_{X,0} - X_0 F^T_{X,0} X_0)$; $H_0 = -G_0$;
3:  if ($curve == geod$) then
4:    $A_0 = X_0^T H_0$;
5:    Compute $A_{0,2} = A_0 A_0$, $A_{0,4} = A_{0,2} A_{0,2}$, $A_{0,6} = A_{0,2} A_{0,4}$ and $n_1 A_0 = \|A_0\|_1$;
6:  for $k = 0, 1, 2, \dots$ do
7:    $[t_{k+1}, E_{t_{k+1}}] = \text{linesearch}(F_\Gamma, F_0, n_1 A_k, A_k, A_{k,2}, A_{k,4}, A_{k,6}, B^{(p)}_k)$;
8:    if ($curve == geod$) then
9:      $X_{k+1} = X_k \cdot E_{t_{k+1}}$;        /* $E_{t_{k+1}} = e^{t_{k+1} A_k}$ */
10:     $\Xi_{k+1} = \Pi(H_k) = X_{k+1} \cdot A_k$;
11:   else
12:     $X_{k+1} = E_{t_{k+1}}$;        /* $E_{t_{k+1}}$ is Q factor of $X_k + t_{k+1} H_k$ */
13:     $\Xi_{k+1} = \frac{1}{2} X_{k+1}(X^T_{k+1} H_k - H^T_k X_{k+1})$;
14:   Compute $A^{(p)}_{X,k+1} = A^{(p)} X_{k+1}$; $B^{(p)}_{k+1} = (A^{(p)}_{X,k+1})^T X_{k+1}$, for $p = 1, \dots, m$;
15:   $F_{X,k+1} = F_X(A^{(p)}_{X,k+1}, B^{(p)}_{k+1})$; $F_{k+1} = F(B^{(p)}_{k+1})$; $G_{k+1} = \frac{1}{2}(F_{X,k+1} - X_{k+1} F^T_{X,k+1} X_{k+1})$;
16:   if ($conj == exact$) then
17:     $I_{k+1} = (F^T_{X,k+1} X_{k+1} + X^T_{k+1} F_{X,k+1}) \Xi^T_{k+1}$;
18:     $J_{k+1} = F_{XX}(A^{(p)}, A^{(p)}_{X,k+1}, B^{(p)}_{k+1}, G_{k+1}, \Xi_{k+1}) - \frac{1}{2}\operatorname{tr}(I_{k+1} G_{k+1})$;
19:     $K_{k+1} = F_{XX}(A^{(p)}, A^{(p)}_{X,k+1}, B^{(p)}_{k+1}, \Xi_{k+1}, \Xi_{k+1}) - \frac{1}{2}\operatorname{tr}(I_{k+1} \Xi_{k+1})$;
20:   if ($conj = PR$) then
21:     if ($curve == geod$) then
22:       $E = ss\_alg\_exp(t/2, n, n1Ak, Ak, A_{k,2}, A_{k,4}, A_{k,6})$;
23:       $\Pi(G_k) = X_k E X^T_k G_k E$        /* $\Pi(G_k) = X_k e^{\frac{t_{k+1} A_k}{2}} (X^T_k G_k) e^{\frac{t_{k+1} A_k}{2}}$ */
24:     else
25:       $\Pi(G_k) = \frac{1}{2} X_{k+1}(X^T_{k+1} G_k - G^T_k X_{k+1})$
26:     $W_{k+1} = G_{k+1} - \Pi(G_k)$
27:     $J_{k+1} = \operatorname{tr}(W^T_{k+1} \cdot G_{k+1})$;
28:     $K_{k+1} = \operatorname{tr}(G^T_k \cdot G_k)$;
29:   if ($conj == FR$) then
30:     $J_{k+1} = \operatorname{tr}(G^T_{k+1} \cdot G_{k+1})$;
31:     $K_{k+1} = \operatorname{tr}(G^T_k \cdot G_k)$;
32:   $\beta_{k+1} = \frac{J_{k+1}}{K_{k+1}}$;
33:   $H_{k+1} = -G_{k+1} + \beta_{k+1} \Xi_{k+1}$;
34:   if ($curve == geod$) then
35:     $A_{k+1} = X^T_{k+1} H_{k+1}$;
36:     Compute $A_{k+1,2} = A_{k+1} A_{k+1}$, $A_{k+1,4} = A_{k+1,2} A_{k+1,2}$, $A_{k+1,6} = A_{k+1,2} A_{k+1,4}$,
        $n_1 A_{k+1} = \|A_{k+1}\|_1$;
```

An efficient parallel version of the conjugate gradient method on the Stiefel manifold is displayed in Algorithm 1, with a choice for three important elements: *curve*, which determines the choice of line search curve: a geodesic or retraction, *linesearch* for the choice of the line search method, and *conj* for the choice of the conjugacy formula: exact conjugacy, the Polak–Riebière (PR), or the Fletcher–Reeves (FR) formula.

## 3.2. Implementation of the method on the oblique manifold

We were more focused on the optimization algorithm on the oblique manifold, as this problem lacks a compact-form solution algorithm and offers greater opportunities for parallelization.

When computing the gradient and the Riemannian Hessian on the oblique manifold, three distinct operations must be considered: extracting diagonal elements of a matrix product, multiplying a matrix with a diagonal matrix, and finding the trace of such a product. From (17) and (19) we see that the gradient and the Hessian require diagonal elements of the same matrix product: $X^T F_X(X)$. These diagonal elements are computed by $n$ calls of `ddot()` on the columns of the factor matrices, and the set $\{1,\dots,n\}$ is uniformly partitioned across the threads in function `diagonal_of_transpose_product()`.

We implemented a function that computes the elements of $B + A\cdot D$, where $D$ is a diagonal matrix, such that $n^2$ elements are uniformly distributed across the threads in function `product_of_matrix_and_diagonal()`. This function is used to compute $\mathrm{grad}F(X)$, where diagonal elements of $X^T F_X(X)$ are computed for the first time and stored in a vector. The operation $\mathrm{tr}(F_X(X)^T X \mathrm{Diag}(\Xi_1^T \Xi_2))$, required by the evaluation of $\mathrm{Hess}F(X)$, is performed by computing the diagonal elements of $\mathrm{Diag}(\Xi_1^T \Xi_2))$ and storing them in a second vector, and finally by applying `ddot()` to these two vectors. In the case of the oblique manifold, we have no problems with evaluating the geodesic. The form of the geodesic is very simple, it requires only two multiplications of a matrix by a diagonal matrix, and a sum of two matrices. We are going to exploit its simple form in order to obtain an efficient algorithm for the minimization of function $F$ along a geodesic. So, the minimization of $F$ along geodesic $\Gamma_k$ through $\Gamma_k(0) = X_k$ in direction $\dot{\Gamma}_k(0) = H_k$, with $\mathrm{Diag}(X_k^T H_k) = 0$, is again obtained by employing an unconstrained optimization method that does not use derivatives. By direct computation we get

$$\begin{aligned}
F_{\Gamma_k}(t) = F(\Gamma_k(t)) = \frac{1}{2}\sum_{p=1}^{m} \Big[&\mathrm{tr}(C_k(t)^2 D_{k,1}^{(p)}(t)) + 2\,\mathrm{tr}(C_k(t)S_k(t)\Lambda_k^{-1}D_{k,2}^{(p)}(t)) \\
&+ 2\,\mathrm{tr}(C_k(t)^2 D_{k,3}^{(p)}(t)) + 2\,\mathrm{tr}(C_k(t)S_k(t)\Lambda_k^{-1}D_{k,4}^{(p)}(t)) \\
&+ 2\,\mathrm{tr}(C_k(t)S_k(t)\Lambda_k^{-1}D_{k,5}^{(p)}(t)) + \mathrm{tr}(C_k(t)^2 D_{k,6}^{(p)}(t)) \\
&+ \mathrm{tr}(S_k(t)^2\Lambda_k^{-2}D_{k,7}^{(p)}(t)) + 2\,\mathrm{tr}(C_k(t)S_k(t)\Lambda_k^{-1}D_{k,8}^{(p)}(t)) \\
&+2\,\mathrm{tr}(S_k(t)^2\Lambda_k^{-2}D_{k,9}^{(p)}(t)) + \mathrm{tr}(S_k(t)^2\Lambda_k^{-2}D_{k,10}^{(p)}(t))\Big],
\end{aligned} \tag{22}$$

$$\begin{aligned}
D_{k,1}^{(p)}(t) &= \text{Diag}\,(oB_k^{(p)}E_{k,1}(t)oB_k^{(p)}) & D_{k,2}^{(p)}(t) &= \text{Diag}\,(oB_k^{(p)}E_{k,1}(t)oB_{k,1}^{(p)})\\
D_{k,3}^{(p)}(t) &= \text{Diag}\,(oB_k^{(p)}E_{k,2}(t)(oB_{k,1}^{(p)})^T) & D_{k,4}^{(p)}(t) &= \text{Diag}\,(oB_k^{(p)}E_{k,2}(t)oB_{k,2}^{(p)})\\
D_{k,5}^{(p)}(t) &= \text{Diag}\,(oB_{k,1}^{(p)}E_{k,2}(t)oB_{k,1}^{(p)}) & D_{k,6}^{(p)}(t) &= \text{Diag}\,(oB_{k,1}^{(p)}E_{k,3}(t)(oB_{k,1}^{(p)})^T)\\
D_{k,7}^{(p)}(t) &= \text{Diag}\,((oB_{k,1}^{(p)})^TE_{k,1}(t)oB_{k,1}^{(p)}) & D_{k,8}^{(p)}(t) &= \text{Diag}\,(oB_{k,1}^{(p)}E_{k,3}(t)oB_{k,2}^{(p)})\\
D_{k,9}^{(p)}(t) &= \text{Diag}\,((oB_{k,1}^{(p)})^TE_{k,2}(t)oB_{k,2}^{(p)}) & D_{k,10}^{(p)}(t) &= \text{Diag}\,(oB_{k,2}^{(p)}E_{k,3}(t)oB_{k,2}^{(p)}) \quad (23)
\end{aligned}$$

where $\Lambda_k = \text{diag}\,(\|H_k(:,1)\|_2, \dots, \|H_k(:,n)\|_2)$, and

$$\begin{aligned}
oB_k^{(p)} &= \text{Off}(X_k^TA^{(p)}X_k) & C_k(t) &= \cos(\Lambda_k t) & E_{k,1}(t) &= C_k(t)^2\\
oB_{k,1}^{(p)} &= \text{Off}(X_k^TA^{(p)}H_k) & S_k(t) &= \sin(\Lambda_k t) & E_{k,2}(t) &= C_k(t)S_k(t)\Lambda_k^{-1}\\
oB_{k,2}^{(p)} &= \text{Off}(H_k^TA^{(p)}H_k) & & & E_{k,3}(t) &= S_k(t)^2\Lambda_k^{-2}.
\end{aligned}$$

Hence, the evaluation of $F(\Gamma_k(t))$ reduces to summing up the traces of diagonal matrix products. When evaluating this function for different parameters $t$ in the line-search optimization method, the products $oB_k^{(p)}$, $oB_{k,1}^{(p)}$ and $oB_{k,2}^{(p)}$ are computed only once using `multiple_gemms()`, and only the diagonal matrices $C_k(t)$ and $S_k(t)$ are evaluated each time. $C_k(t)$ and $S_k(t)$ are computed in parallel, so that $n$ diagonal elements are uniformly distributed across the threads, together with the diagonal elements of the matrices $E_{k,1}(t)$, $E_{k,2}(t)$, and $E_{k,3}(t)$. These diagonal matrices appear as factors of $D_{k,i}^{(p)}(t)$, where $p = 1, \dots, m$ and $i = 1, \dots, 10$, and as factors of terms in (22). Further, only diagonal elements of $D_{k,i}^{(p)}(t)$ are computed in a similar way as before when diagonal elements were extracted from a matrix product in `diagonal_of_transpose_product()`. Computation of a total of $10mn$ diagonal elements is uniformly distributed across the threads in function `compute_diag_D()_geod`. Finally, $10m$ traces are uniformly distributed across the threads in function `compute_sum_traces_for_F_geod()`, and computed in parallel by applying the `ddot()` function to two vectors storing diagonals of a $D_{k,i}^{(p)}(t)$ matrix and an $E_{k,j}(t)$ matrix.

We can conclude that for every new $t$, the evaluation of $F(\Gamma_k(t))$ requires $\mathcal{O}(mn^2)$ instead of $\mathcal{O}(mn^3)$ for direct computation of

$$2\sum_{p=1}^{m} \text{tr}(\text{Off}(\Gamma_k(t)^TA^{(p)}\Gamma_k(t))\Gamma_k(t)^TA^{(p)}\dot{\Gamma}_k(t))$$

by first computing $\Gamma_k(t)$ and by applying matrix multiplications. Again, the function that implements the evaluation of $F(\Gamma_k(t))$, as described, is denoted by $F_\Gamma$, and it is implemented by the function `F_geod()`.

As in the case with the Stiefel manifold, the parallel translation of the search direction $H_k$ on the oblique manifold reduces to the derivative of the geodesic $\Gamma_k$, so that

$$\Pi(H_k) = \dot{\Gamma}_k(t_{k+1}) = -X_k\Lambda_k\sin(\Lambda_k t_{k+1}) + H_k\cos(\Lambda_k t_{k+1}).$$

Computation of $\Pi(H_k)$ is as simple as computation of a point on the geodesic.

In the case when line search is performed along the retraction going through $\Gamma_k(0) = X_k$ in direction $\dot{\Gamma}_k(0) = H_k$, with $\mathrm{Diag}(X_k^T H_k) = 0$, direct computation reveals a similar structure of $F(\Gamma_k(t))$ to the same composition of functions when $\Gamma_k$ is a geodesic. First, we can observe that computing elements of the diagonal matrices $N_k(t)$ and $M_k(t)$, defined in (13) and (15), can be simplified as

$$N_k(t) = (I + t^2 \Lambda_k t)^{\frac{1}{2}} \qquad M_k(t) = tM_k, \tag{24}$$

and $M_k$ is defined in (11). In this case, we have

$$\begin{aligned} F_{\Gamma_k}(t) = F(\Gamma_k(t)) = \frac{1}{2}\sum_{p=1}^{m} \Big[ &\mathrm{tr}(N_k(t)^{-2} D_{k,1}^{(p)}(t)) + 2t\,\mathrm{tr}(N_k(t)^{-2} D_{k,2}^{(p)}(t)) \\ &+ 2t\,\mathrm{tr}(N_k(t)^{-2} D_{k,3}^{(p)}(t)) + 2t^2\,\mathrm{tr}(N_k(t)^{-2} D_{k,4}^{(p)}(t)) \\ &+ 2t^2\,\mathrm{tr}(N_k(t)^{-2} D_{k,5}^{(p)}(t)) + t^2\,\mathrm{tr}(N_k(t)^{-2} D_{k,6}^{(p)}(t)) \\ &+ t^2\,\mathrm{tr}(N_k(t)^{-2} D_{k,7}^{(p)}(t)) + 2t^3\,\mathrm{tr}(N_k(t)^{-2} D_{k,8}^{(p)}(t)) \\ &+ 2t^3\,\mathrm{tr}(N_k(t)^{-2} D_{k,9}^{(p)}(t)) + t^4\,\mathrm{tr}(N_k(t)^{-2} D_{k,10}^{(p)}(t)) \Big]. \end{aligned} \tag{25}$$

Matrices $D_{k,i}^{(p)}(t)$, $i = 1, \dots, 10$ also satisfy (23), only $E_{k,1}(t) = E_{k,2}(t) = E_{k,3}(t) = N_k(t)^{-2}$. In this case, we use only one additional vector for storing diagonal elements of $E_{k,1}(t) = N_k(t)^{-2}$, and computation of a total of $10mn$ diagonal elements is uniformly distributed across the threads in function `compute_diag_D()_retr`. Finally, $10m$ traces are uniformly distributed across the threads in function `compute_sum_traces_for_F_retr()`, and are computed in parallel by applying the `ddot()` function to two vectors storing diagonals of a $D_{k,i}^{(p)}(t)$ matrix and the $E_{k,1}(t)$ matrix. The function that implements the evaluation of $F(\Gamma_k(t))$, where $\Gamma_k$ is a retraction, is implemented by the function `F_retr()`.

Vector transport of any vector $\Xi_0$ along retraction $\Gamma_k$ is again as simple as computation of a point on the retraction:

$$\Pi(\Xi_0) = \Xi_0 - t\Gamma_k(t) N_k(t)^{-1} M_k.$$

As we can see, all components of the conjugate gradient minimization algorithm on the oblique manifold consist of simple matrix operations, completely avoiding solving linear systems. By taking the forms of the objective function and geodesics into account, we also accomplish a reduction in computational complexity.

An efficient parallel version of the conjugate gradient method on the oblique manifold is displayed in Algorithm 2.

## 3.3. Parallel implementations

Let us summarize all the techniques used in our parallel implementations of the conjugate gradient algorithm on two matrix manifolds. The building blocks of the algorithms are reduced to matrix multiplications and sums, computation of diagonal elements or evaluation of traces on matrix products, and evaluation of the matrix

**Algorithm 2** Parallel CG on the oblique manifold

```
Require: functions $F$, $F_X$, $F_{XX}$, $F_\Gamma(t) = F(\Gamma(t))$, $X_0 \in \mathcal{O}_{n,n}$, parameters curve indicating
    choice of line search curve, linesearch indicating line search method, conj indicating conjugacy
    formula.
 1: Compute $A^{(p)}_{X,0} = A^{(p)} X_0$; $B^{(p)}_0 = (A^{(p)}_{X,0})^T X_0$, for $p = 1, \dots, m$;
 2: $F_{X,0} = F_X(A^{(p)}_{X,0}, B^{(p)}_0)$; $F_0 = F(B^{(p)}_0)$; $G_0 = F_{X,0} - X_0 \mathrm{Diag}(X_0^T F_{X,0})$;
 3: $H_0 = -G_0$; $\Lambda_0 = \mathrm{diag}\ (\|(H_0)_1\|_2, \dots, \|(H_0)_n\|_2)$;
 4: Compute $B^{(p)}_{0,1} = X_0^T A^{(p)} H_0$, $B^{(p)}_{0,2} = H_0^T A^{(p)} H_0$, for $p = 1, \dots, m$;
 5: for $k = 0, 1, 2, \dots$ do
 6:     $t_{k+1} = \mathrm{linesearch}(F_\Gamma, F_0, \Lambda_k, B^{(p)}_k, B^{(p)}_{k,1}, B^{(p)}_{k,2})$;
 7:     Compute vectors $C_k(t_{k+1})$ and $S_k(t_{k+1})$, or $N_k(t_{k+1})$;
 8:     if (curve == geod) then
 9:         $\Xi_{k+1} = \Pi(H_k) = -X_k \Lambda_k S_k(t_{k+1}) + H_k C_k(t_{k+1})$;
10:         $X_{k+1} = X_k C_k(t_{k+1}) + H_k \Lambda_k^{-1} S_k(t_{k+1})$;
11:     else
12:         $X_{k+1} = (X_k + t_{k+1} H_k N_k(t_{k+1})^{-1}$;
13:         $\Xi_{k+1} = H_k - t_{k+1} X_{k+1} \Lambda_k^2 N_k(t_{k+1})^{-1}$;        /* $M_k = \Lambda_k^2$ */
14:     Compute $A^{(p)}_{X,k+1} = A^{(p)} X_{k+1}$; $B^{(p)}_{k+1} = (A^{(p)}_{X,k+1})^T X_{k+1}$, for $p = 1, \dots, m$;
15:     $F_{X,k+1} = F_X(A^{(p)}_{X,k+1}, B^{(p)}_{k+1})$; $F_{k+1} = F(B^{(p)}_{k+1})$;
16:     $I_{k+1} = \mathrm{Diag}(F^T_{X,k+1} X_{k+1})$; $G_{k+1} = F_{X,k+1} - X_{k+1} \cdot I_{k+1}$;
17:     if (conj == exact) then
18:         $J_{k+1} = F_{XX}(A^{(p)}, B^{(p)}_{k+1}, G_{k+1}, \Xi_{k+1}) - \mathrm{tr}(I_{k+1} \mathrm{Diag}(G^T_{k+1} \Xi_{k+1}))$;
19:         $K_{k+1} = F_{XX}(A^{(p)}, B^{(p)}_{k+1}, \Xi_{k+1}, \Xi_{k+1}) - \mathrm{tr}(I_{k+1} \mathrm{Diag}(\Xi^T_{k+1} \Xi_{k+1})$;
20:     if (conj = PR) then
21:         $I_{k+1} = \mathrm{diag}(H_k(:,1)^T G_k(:,1), \dots, H_k(:,n)^T G_k(:,n))$;        /* $I_{k+1} = M_k$ */
22:         if (curve == geod) then
23:             $\Pi(G_k) = G_k - X_k S_k(t_{k+1}) I_{k+1} \Lambda_k^{-1} + H_k(C_k(t_{k+1}) - Id_n) I_{k+1} \Lambda_k^{-2}$
24:         else
25:             $\Pi(G_k) = G_k - t_{k+1} X_{k+1} I_{k+1} N_k(t_{k+1})^{-1}$;
26:         $W_{k+1} = G_{k+1} - \Pi(G_k)$
27:         $J_{k+1} = \mathrm{tr}(W^T_{k+1} \cdot G_{k+1})$;
28:         $K_{k+1} = \mathrm{tr}(G_k^T \cdot G_k)$;
29:     if conj == FR then
30:         $J_{k+1} = \mathrm{tr}(G^T_{k+1} \cdot G_{k+1})$;
31:         $K_{k+1} = \mathrm{tr}(G_k^T \cdot G_k)$;
32:     $\beta_{k+1} = \frac{J_{k+1}}{K_{k+1}}$;
33:     $H_{k+1} = -G_{k+1} + \beta_{k+1} \Xi_{k+1}$; $\Lambda_{k+1} = \mathrm{diag}\ (\|(H_{k+1})_1\|_2, \dots, \|(H_{k+1})_n\|_2)$;
34:     Compute $B^{(p)}_{k+1,1} = X^T_{k+1} A^{(p)} H_{k+1}$, $B^{(p)}_{k+1,2} = H^T_{k+1} A^{(p)} H_{k+1}$, for $p = 1, \dots, m$;
```

exponential only for the Stiefel manifold. Most of the products and traces involving input matrices $A^{(p)}$ are computed in parallel. Further, some repeating products are computed once, stored, and reused. When computing traces of matrix products in form $\mathrm{tr}(M_1 \cdot M_2)$ or $\mathrm{tr}(M_1^T \cdot M_2)$, or when extracting diagonals of matrix products in form $D_\ell = \mathrm{Diag}\ (M_{1,\ell} \cdot L_\ell \cdot M_{2,\ell})$, for diagonal matrices $L_\ell$, only diagonal elements of the products are computed in parallel, mostly by using parallel calls to the `ddot()` function. The `ddot()` function is also used for computing $\mathrm{tr}(L_i \cdot M_j)$, where $L_i$ and $M_j$ are diagonal matrices stored as vectors. The line search method used for finding a local minimum of $F(\Gamma_k(t))$ performs only minimum changes for different parameters $t$, reducing the computational complexity of this task.

It only remains to say a few words about actual implementation of the algorithms on the specific computing platform. We produced several versions of both conjugate gradient algorithms, trying to find the most efficient approach. Our parallel implementations are designed for multi-core systems, where we explicitly handled tasks for each thread by using the POSIX thread library. We also used multi-threaded BLAS functions, mostly `dgemm()` and `ddot()`, where we controlled the number of threads used by BLAS functions. When multiple calls to a BLAS function were executed at once, each one in its own thread, then the number of calling threads multiplied by the number of BLAS threads was equal to the maximum number of threads supported by our system, which was 24.

## 4. Computational complexity

The most complex operations in all algorithm variants are those involving matrix products. These operations are mostly concentrated in the functions for computing $F(\Gamma(t))$, $F_X(X)$ and $F_{XX}(X)(\Xi_1, \Xi_2)$. Since we developed many variants of the conjugated gradient method on two manifolds, we are going to present complexity analysis results for the variants with the most complex curve (geodesic) and conjugacy (exact) computations, employing Armijo's backtracking method for line search on both manifolds. The numerical experiments revealed that this was the optimal variant. Table 1 displays the results of the analysis.

| Alg. | $F(\Gamma(t))$ | $F_X(X))$ | $F_{XX}(X)(\Xi_1, \Xi_2)$ | CG |
|---|---|---|---|---|
| OS | $6mn^3$ | $6mn^3$ | $18mn^3$ | $[10 + i(6i_a + 52)]mn^3$ |
| PS | $\frac{4mn^3}{p}$ | $\frac{6mn^3}{p}$ | $\frac{8mn^3}{p}$ | $[6 + i(4i_a + 26)]\frac{mn^3}{p}$ |
| OO | $4mn^3$ | $6mn^3$ | $18mn^3$ | $[10 + i(4i_a + 50)]mn^3$ |
| PO | $\frac{20mn^2}{p}$ | $\frac{6mn^3}{p}$ | $\frac{8mn^3}{p}$ | $[12 + 20i]\frac{mn^3}{p}$ |

Table 1: *Results of the complexity analysis for the most complex operations and the whole CG algorithm up to $\mathcal{O}(i \cdot i_a(m+n)n^2/p)$. Here, $i$ is the number of CG iterations, $i_a$ is the number of Armijo's backtracking iterations, and $p$ is number of threads. OS, PS, OO and PO stand for the original CG algorithm on the Stiefel manifold, its parallel version, the original CG algorithm on the oblique manifold, and its parallel version, respectively.*

The result shows that parallel versions of the conjugate gradient method on both manifolds, executed on only one thread, are less complex than the original versions. This is a consequence of the modifications and operation reorganizations. In particular, we can notice that the modified computation of $F(\Gamma(t))$ on the oblique manifold, described in subsection 3.2, indeed reduced its complexity to $\mathcal{O}(mn^2)$.

## 5. Numerical experiments

Our numerical experiments were performed on the computational environment consisting of two Intel(R) Xeon(R) E5-2690 v3 processors (2.60 GHz) with 24 cores in total, where each processor is equipped with 30 MB of cache memory, 256 GB RAM, and Intel Parallel Studio XE 2016 + MKL 11.3 was used for program compilations. The CPUs reach the peak `dgemm()` performance of about 800 Gflops.

First, variants of the conjugate gradient method on both manifolds were implemented in two forms:

- the original form, as described in Section 2, is generic; it does not exploit the form of the objective function and specific geometric properties of the manifolds, and any other objective function or manifold can be used instead of our specific choices; in terms of the implementation type, this form is sequential and implicitly parallel only via multi-threaded BLAS;
- the parallel and modified form, as described in Algorithms 1 and 2, implements our techniques described in Section 3; this form exploits specific forms of the objective function and manifolds, and is generally multi-threaded.

Further, we performed two rounds of tests for all conjugate gradient variants and forms on both manifolds. In the first round, we tested only the speedup obtained by parallel forms of the algorithm variants when compared to their original forms. Hence, we compared execution times only for one iteration of each algorithm variant. Speedup factors are computed as $t_o/t_p$, where $t_o$ is the execution time of the original version, and $t_p$ is the time of the parallel version executed on $p$ threads. Then, in the second round, we performed the convergence test, where all the variants in both forms on the specific manifolds were executed with the same input parameters and matrices, and with the same stopping criteria. We were interested in finding the most efficient variant in terms of the execution time.

In the first test round, we tested our implementations on a large number of examples with different matrix dimensions and different numbers of input matrices. The dimensions of the matrices were taken from the set $n \in \{10, 100, 1000, 2000\}$, and the number of input matrices were chosen from $m \in \{10, 20, 50, 100, 200, 1000\}$. For each choice of $n$ and $m$, we generated $m$ input matrices $A^{(p)} = Y^{-T} \cdot D^{(p)} \cdot Y^{-1} \in \mathbb{R}^{n\times n}$, with random matrix $Y$ from the observed manifold. Diagonal elements of the diagonal matrices $D^{(p)}$ are chosen as $d_{ii}^{(p)} = (-1)^p(i + p \cdot b)$, where $b = 10$ in the first test round and $b = 10^{-3}$ in the second.

As mentioned before, the algorithm variants are determined by three parameters, i.e. *curve*, *linesearch* and *conj*. The choices are as follows:

- *curve* – a geodesic or retraction,
- *linesearch* – the Nelder–Mead method or Armijo's backtracking,
- *conj* – exact conjugacy, the Polak–Riebière (PR) formula, or the Fletcher–Reeves (FR) formula.

Both line search methods use only function values, and the computation of derivatives is not required. The Nelder-Mead method computes the local minimum up to a given tolerance on the accuracy. Armijo's backtracking is a common choice for finding a sub-optimal solution very efficiently, but on the other hand, this line search method can still guarantee the convergence of the conjugate gradient method. Altogether, we implemented 12 versions of each algorithm form on the specific manifold.

Before executing our two test rounds, we had to determine the optimal number of BLAS threads used in functions `traces_by_ddots()` and `multiple_gemms()`. We tested these two functions for the same ranges of $n$ and $m$ as for the conjugate gradient algorithms, and the number of BLAS threads was chosen from the set $nbt \in \{24, 12, 8, 6, 4, 3, 2, 1\}$. The number of calling threads was taken as $24/nbt$. Each function was tested with two possible choices of the *trans* parameter for the first factor matrix. The optimal numbers of BLAS threads obtaining the lowest execution time, are stored for later use. In all our further tests, as the execution time of CG variants in parallel form for each $n$ and $m$, we took the best among the timings obtained by taking the optimal numbers of BLAS threads.

The second test round tested convergence of all variants on specific manifolds, with the same input matrices where $n = 100$ and $m = 1000$, and with the same $X_0$. For simplicity of the convergence control, we chose diagonal input matrices. The stopping criteria were also the same for all variants, specifying that iterations should stop when $F(X_k) < 10^{-5}F(X_0)$.

Codes written in the C language for both test rounds are available at: `https://github.com/NelaBosner/CG_JAD`.

## 5.1. Results of the speedup tests

The obtained timings of all algorithm forms and variants on both manifolds show that the most time-consuming variants, in original and in parallel forms, are the ones implementing Nelder–Mead method for line search. Such variants where line search is performed on retractions are noticeably faster than those using geodesics, but only in their original form. Choosing approximate conjugacy formulas only slightly speeds up the execution time. The fastest iterations are obtained by variants implementing Armijo's backtracking. Hence, we can conclude that the choice of the line search method has the biggest influence on the execution time of one iteration, variants that use retractions are slightly faster, and the choice of the conjugacy formula has only a minor influence. Due to the lower computation complexity of the conjugate gradient methods on the oblique manifold, the execution times are lower than those for the algorithms on the Stiefel manifold, especially for larger numbers of input matrices.

Speedup factors obtained by comparing one iteration of the parallel CG algorithms on the Stiefel manifold with one iteration of its original form are displayed in Figure 1. We can conclude that the largest speedup factors are obtained for the most expensive original versions, and that they grow with the number of input matrices. It is also obvious that it does not pay off to apply the parallel algorithm to the input matrices of low dimension, such as $n = 10$, or when the number of input matrices $m$ is small. That is no surprise, since we were focused on the larger dimensions that can exploit the full potential of parallelism. On the other hand, the speedup factors do not monotonically increase as n increases, because we compare our parallel implementations with the original algorithms which are not completely sequential. They use multi-threaded BLAS functions which are more efficient for larger dimensions. Speedup factors for larger dimensions and a larger number of input matrices range between 1.6 and 5.8.

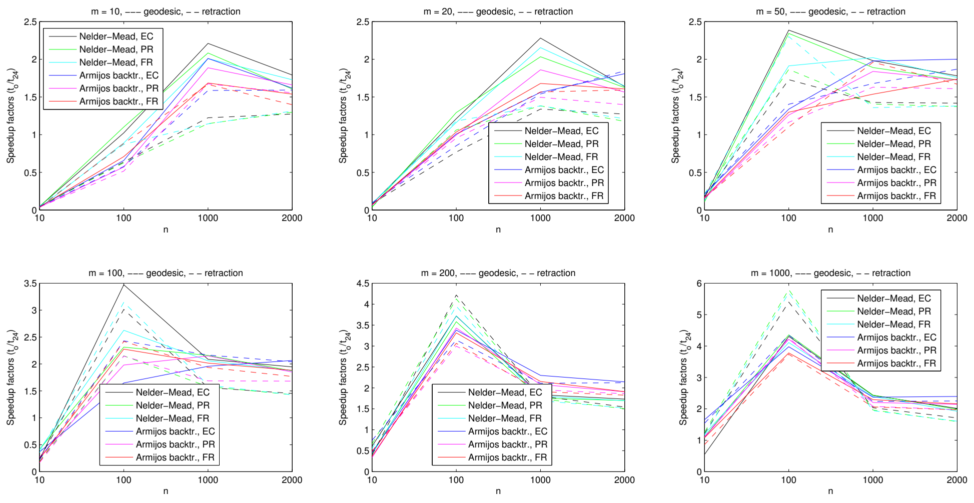


Figure 1: *Speedup factors obtained for one iteration of the parallel CG algorithms on the Stiefel manifold compared to its original form, with different numbers of input matrices $m$. Full lines represent variants with line search performed on geodesics, and dashed lines those with line search on retractions.*

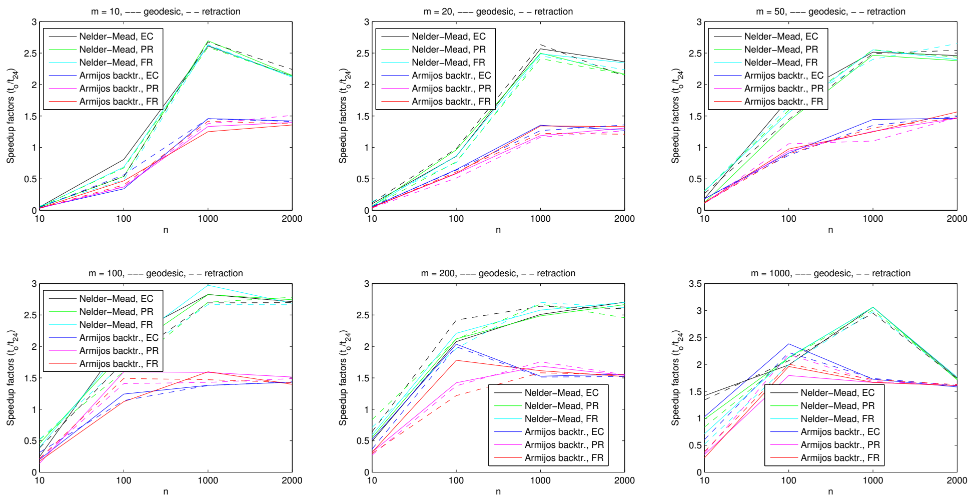
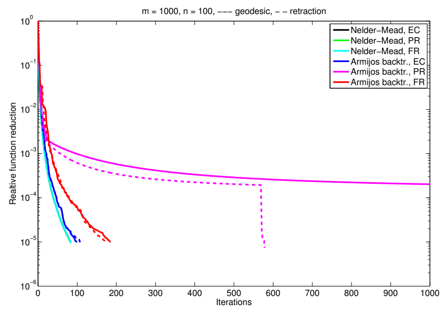


Figure 2: *Speedup factors obtained for one iteration of the parallel CG algorithms on the oblique manifold compared to its original form, with different numbers of input matrices $m$. Full lines represent variants with line search performed on geodesics, and dashed lines those with line search on retractions.*

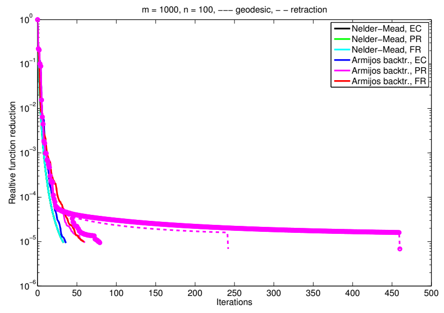
Similar results are obtained for parallel CG algorithms on the oblique manifold, where the lower complexity of these algorithms also influences the speedup factors, which are displayed in Figure 2. They are higher than those for the Stiefel manifold when the number of input matrices $m$ is smaller, and lower for larger $m$-s. Speedup factors for larger dimensions and a larger number of input matrices range between 1.46 and 3.06.

## 5.2. Results of the convergence tests

The best insight into the efficiency of all versions is given in the results of the second test round. First, we were interested in the number of iterations required to reach the stopping criteria, in order to see which conditions affect the most possible prolonged convergence. Figure 3 plots relative objective function reduction against the number of iterations for all versions of the CG algorithms.



(a) *CG on the Stiefel manifold.*



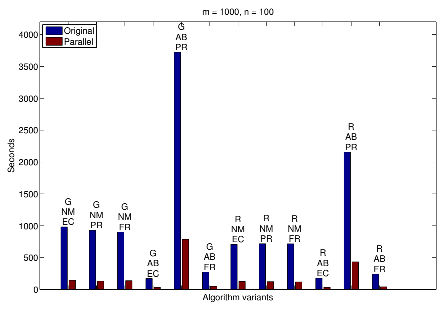
(b) *CG on the oblique manifold.*

Figure 3: *Relative objective function reduction measured as* $F(X_k)/F(X_0)$ *against the number of iterations until it reaches stopping criteria* $10^{-5}$. *Full lines represent variants with line search performed on geodesics, and dashed lines those with line search on retractions. Where the parallel and the original versions give the same result, only the results of the original version are presented. The parallel versions are denoted by circles.*

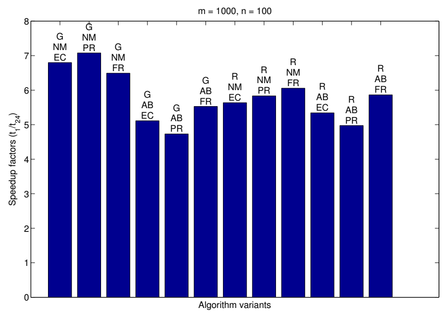
The magenta line in this figure represents all variants that implement the Nelder–Mead method for line search. They all obtained the minimum number of iterations, and this is consistent with the recommendation for the exact line search in [4]. On the other hand, a suboptimal solution of line search obtained by Armijo's backtracking in combination with the approximate conjugacy formulas prolongs the convergence. Especially, combination with the Polak–Riebière formula on the Stiefel manifold using geodesics did not reach the given stopping criteria even after 1000 iterations.

Finally, the most interesting data are total execution times required to reach stopping criteria, displayed in Figures 4a and 5a for the Stiefel manifold and for the oblique manifold, respectively.

As we can see, the fastest version in both its original and parallel forms on both manifolds is the one that performs line search employing Armijo's backtracking and uses the exact conjugacy formula. For the versions in their original form on the Stiefel manifold, and for both forms on the oblique manifold, the one that performs line search on a geodesic is the fastest. Finally, we can conclude that the version that implements Armijo's backtracking and the exact conjugacy formula in the parallel form is by far the fastest and the most efficient algorithm for our problem, requiring

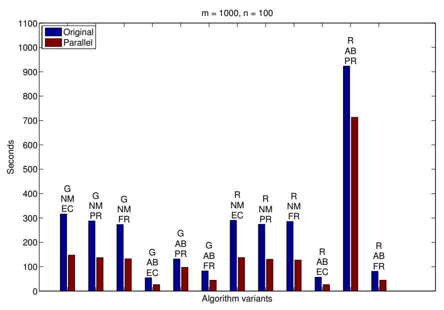


(a) *Total execution times.*

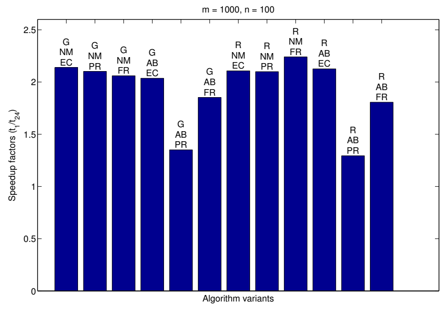


(b) *Speedup factors.*

Figure 4: *Total execution times and speedup factors obtained for reaching the same stopping criteria on the Stiefel manifold. Notation: G = geodesic, R = retraction, NM = Nelder–Mead method, AB = Armijo's backtracking, EC = exact conjugacy, PR = Polak–Riebière formula, FR = Fletcher–Reeves formula.*



(a) *Total execution times.*



(b) *Speedup factors.*

Figure 5: *Total execution times and speedup factors obtained for reaching the same stopping criteria on the oblique manifold. Notation: G = geodesic, R = retraction, NM = Nelder–Mead method, AB = Armijo's backtracking, EC = exact conjugacy, PR = Polak–Riebière formula, FR = Fletcher–Reeves formula.*

only about 33 seconds to finish the job for 1000 input matrices of dimension $100 \times 100$ on the Stiefel manifold, and only about 27 seconds to complete the same task on the oblique manifold. Numerical results also show that there is no reason to avoid computing the Hessian and the geodesic, and that a combination of inexact line search solving with approximate conjugacy formulas does not produce the most efficient algorithm.

The speedup factors for the second test round are presented in Figure 4b for the Stiefel manifold and Figure 5b for the oblique manifold. These speedup factors are even larger than those obtained for a single iteration. This means that we succeeded in making the conjugate gradient method on the Stiefel and the oblique manifolds much more efficient than their original forms.

## 5.3. Scalability

In order to study the strong and weak scalability, we executed one iteration of the optimal algorithm variants that use geodesics and exact conjugacy, and apply Armijo's backtracking for line search, on a different number of threads. The numbers of threads were chosen from the set $p \in \{1, 2, 3, 4, 6, 8, 12, 16, 24\}$. Figures 6 and 7 show the results for the Stiefel manifold and for the oblique manifold, respectively.

In the figures, the results are shown only for specific choices of parameters. In

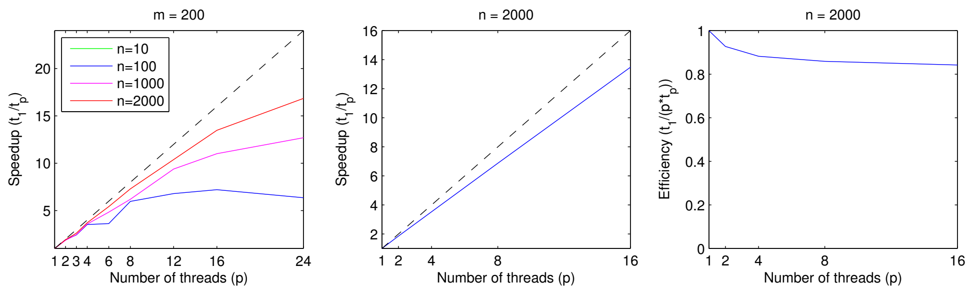


Figure 6: *Strong, weak scalability, and the efficiency of the parallel CG algorithms on the Stiefel manifold. The left subfigure displays speedup factors for* $m = 200$ *and all choices of* $n$*, depending on the number of threads (strong scalability). The middle subfigure also displays speedup factors for* $n = 2000$*, but depending on both* $p$ *and* $m$ *(week scalability), where the ticks on the abscissa stand for* $(p, m)$ *pairs* $\{(1, 10), (2, 20), (4, 50), (8, 100), (16, 200)\}$*. The right subfigure shows the efficiency for the same data as in the middle subfigure.*

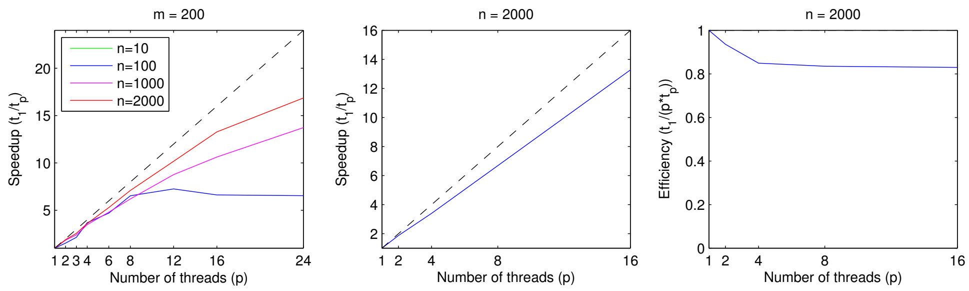


Figure 7: *Strong, weak scalability, and the efficiency of the parallel CG algorithms on the oblique manifold. The left subfigure displays speedup factors for* $m = 200$ *and all choices of* $n$*, depending on the number of threads (strong scalability). The middle subfigure also displays speedup factors for* $n = 2000$*, but depending on both* $p$ *and* $m$ *(week scalability), where the ticks on the abscissa stand for* $(p, m)$ *pairs* $\{(1, 10), (2, 20), (4, 50), (8, 100), (16, 200)\}$*. The right subfigure shows the efficiency for the same data as in the middle subfigure.*

the case of strong scalability $m = 200$, is fixed. Since the computational complexity depends linearly on $m$, $n = 2000$ for weak scalability is fixed, the number of threads is chosen to be power of 2: $\{1, 2, 4, 8, 16\}$, and $m$ is chosen from the set $\{10, 20, 50, 100, 200\}$ where almost all $m$-s are obtained by doubling the previous value. The figures show that strong scalability is the best for larger dimensions, as expected, while weak scalability is quite good, with corresponding efficiency $t_1/(t_p{\cdot}p)$ above 80%.

## 6. Conclusion

In this paper, we propose optimization algorithms for solving two variants of the joint approximate diagonalization problem, with orthogonal and non-orthogonal diagonalizing matrices. The algorithms are based on the conjugate gradient method on two matrix manifolds, where the objective function and the matrix manifolds are chosen to enable efficient parallel implementation. As a result, we developed many variants of parallel implementations for the algorithm on both manifolds. Numerical experiments confirmed that parallel implementations of the conjugate gradient method are more efficient than the original versions. The parallel algorithms on the Stiefel manifold are up to 5.8 times faster than their original forms per iteration, while the parallel algorithms on the oblique manifold achieved the largest speedup factor of 3.06. Additionally, we recommend the most efficient variants of the algorithm on both manifolds: the versions that implement Armijo's backtracking as the line search method and use the exact conjugacy formula in the parallel form. Numerical results also show that there is no reason to avoid computing the Hessian and the geodesic. In the future, we plan to consider GPU implementation of the parallel CG method on the oblique manifold, and further improve its speedup factors, as its versions are particularly well suitable for that purpose.

## Acknowledgement

The author would like to thank the referees for their helpful suggestions and interesting comments.

## References

[1] T. Abrudan, J. Eriksson, V. Koivunen, *Conjugate gradient algorithm for optimization under unitary matrix constraint*, Signal Process. **89**(2009), 1704–1714.

[2] P.-A. Absil, C. G. Baker, K. A. Gallivan, *Trust-region methods on Riemannian manifolds*, Found. Comput. Math. **7**(2007), 303–330.

[3] P.-A. Absil, K. Gallivan, *Joint diagonalization on the oblique manifold for independent component analysis*, in: *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, Vol. **5**, The Institute of Electrical and Electronics Engineers, 2006, pp. V-V.

[4] P.-A. Absil, R. Mahony, R. Sepulchre, *Optimization algorithms on matrix manifolds*, Princeton University Press, Princeton, NJ, 2008.

[5] B. Afsari, *Simple LU and QR based non-orthogonal matrix joint diagonalization*, in: *Independent component analysis and blind signal separation. 6th international conference, ICA 2006, Proceedings*, (J. Rosca, D. Erdogmus and J. C. Principe, Eds.), Simon Haykin Springer, Berlin, 2006, 1–7.

[6] B. Afsari, P. Krishnaprasad, *A novel non-orthogonal joint diagonalization cost function for ICA*, tech. report, Institute for Systems Research, University of Maryland, College Park, MD, 2005.

[7] K. Alyani, M. Congedo, M. Moakher, *Diagonality measures of Hermitian positive-definite matrices with application to the approximate joint diagonalization problem*, Linear Algebra Appl. **528**(2017), 290–320.

[8] *BLAS (Basic Linear Algebra Subprograms)*, `https://www.netlib.org/blas/`.

[9] F. Bouchard, B. Afsari, J. Malick, M. Congedo, *Approximate joint diagonalization with Riemannian optimization on the general linear group*, SIAM J. Matrix Anal. Appl. **41**(2020), 152–170.

[10] F. Bouchard, J. Malick, M. Congedo, *Approximate joint diagonalization according to the natural Riemannian distance*, Lecture Notes in Computer Science **10169**(2017), 290–299.

[11] F. Bouchard, J. Malick, M. Congedo, *Riemannian optimization and approximate joint diagonalization for blind source separation*, IEEE Trans. Signal Process. **66**(2018), 2041–2054.

[12] A. Bunse-Gerstner, R. Byers, V. Mehrmann, *Numerical methods for simultaneous diagonalization*, SIAM J. Matrix Anal. Appl. **14**(1993), 927–949.

[13] J. F. Cardoso, A. Souloumiac, *Blind beamforming for non-gaussian signals*, IEE Proceedings F **140**(1993), 362–370.

[14] J. F. Cardoso, A. Souloumiac, *Jacobi angles for simultaneous diagonalization*, SIAM J. Matrix Anal. Appl. **17**(1996), 161–164.

[15] P. Comon, *Independent component analysis, a new concept?*, Signal Process. **36**(1994), 287–314.

[16] A. Edelman, T. A. Arias, S. T. Smith, *The geometry of algorithms with orthogonality constraints*, SIAM J. Matrix Anal. Appl. **20**(1998), 303–353.

[17] B. N. Flury, W. Gautschi, *An algorithm for simultaneous orthogonal transformation of several positive definite symmetric matrices to nearly diagonal form*, SIAM J. Sci. Stat. Comput. **7**(1986), 169–184.

[18] H. He, D. Kressner, *Randomized joint diagonalization of symmetric matrices*, SIAM J. Matrix Anal. Appl. **45**(2024), 661–684.

[19] N. J. Higham, *The scaling and squaring method for the matrix exponential revisited*, SIAM J. Matrix Anal. Appl. **26**(2005), 1179–1193.

[20] M. Kleinsteuber, H. Shen, *Intrinsic Newton's method on oblique manifolds for overdetermined blind source separation*, in: *Proceedings of the 19th International Symposium on Mathematical Theory of Networks and Systems – MTNS 2010*, (A. Edelmayer, Ed.), Eötvös Loránd University, 2010, 2139–2143.

[21] V. Kuleshov, A. Chaganty, P. Liang, *Tensor Factorization via Matrix Factorization*, in: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, (G. Lebanon and S. V. N. Vishwanathan, Eds.), Proceedings of Machine Learning Research **38**, PMLR, San Diego, 2015, 507–516.

[22] X. Luciani, L. Albera, *Canonical polyadic decomposition based on joint eigenvalue decomposition*, Chemom. Intell. Lab. Syst. **132**(2014), 152–167.

[23] J. H. Manton, *Optimization algorithms exploiting unitary constraints*, IEEE Trans. Signal Process. **50**(2002), 635–650.

[24] O. Micka, A. Weiss, *Estimating frequencies of exponentials in noise using joint diagonalization*, IEEE Trans. Signal Process. **47**(1999), 341–348.

[25] L. Molgedey, H. G. Schuster, *Separation of a mixture of independent signals using time delayed correlations*, Phys. Rev. Lett. **72**(1994), 3634–3637.

[26] D. T. Pham, *Joint approximate diagonalization of positive definite Hermitian matri-*

*ces*, SIAM J. Matrix Anal. Appl. **22**(2001), 1136–1152.

[27] C. Qi, K. A. Gallivan, P.-A. Absil, *Riemannian BFGS Algorithm with Applications*, in: *Recent Advances in Optimization and its Applications in Engineering*, (M. Diehl, F. Glineur, E. Jarlebring, and W. Michiels, Eds.), Springer, Berlin 2010, 183–192.

[28] H. Sato, *Riemannian Newton-type methods for joint diagonalization on the Stiefel manifold with application to independent component analysis*, Optimization **66**(2017), 2211–2231.

[29] H. Sato, *Riemannian optimization and its applications*, SpringerBriefs Electr. Comput. Engin., Springer Cham, Switzerland, 2021.

[30] H. Sato, *Riemannian conjugate gradient methods: general framework and specific algorithms with convergence analyses*, SIAM J. Optim. **32**(2022), 2690–2717.

[31] S. E. Selvan, U. Amato, K. A. Gallivan, C. Qi, M. F. Carfora, M. Larobina, B. Alfano, *Descent algorithms on oblique manifold for source-adaptive ICA contrast*, IEEE Trans. Neural Netw. Learn. Syst. **23**(2012), 1930–1947.

[32] H. Shen, K. Huper, *Block Jacobi-type methods for non-orthogonal joint diagonalisation*, in: *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, The Institute of Electrical and Electronics Engineers, 2009, 3285–3288.

[33] S. T. Smith, *Optimization techniques on Riemannian manifolds*, in: *Hamiltonian and gradient flows, algorithms and control*, (A. Bloch, Ed.), Fields Inst. Commun. **3**, American Mathematical Society, Providence, 1994, 113–136.

[34] R. Vollgraf, K. Obermayer, *Quadratic optimization for simultaneous matrix diagonalization*, IEEE Trans. Signal Process. **54**(2006), 3270–3278.

[35] I. Yamada, T. Ezaki, *An orthogonal matrix optimization by dual Cayley parametrization technique*, in: *Independent component analysis and blind signal separation. 4th international conference, ICA 2003, Proceedings*, (S.-I. Amari, A. Cichocki, S. Makino and N. Murata, Eds.), 2003, pp. 35–40.

[36] A. Yeredor, *Non-orthogonal joint diagonalization in the least-squares sense with application in blind source separation*, IEEE Trans. Signal Process. **50**(2002), 1545–1553.

[37] A. Ziehe, P. Laskov, G. Nolte, K.-R. Müller, *A fast algorithm for joint diagonalization with non-orthogonal transformations and its application to blind source separation*, J. Mach. Learn. Res. **5**(2004), 777–800.