Recurrent Neural Networks Modelling based on Riemannian Symmetric Positive Definite Manifold

Léa Dubreil*†, Samy Labsir*‡, Etienne Rouanet-Labé§, and Gaël Pages†
*TéSA, cooperative research laboratory in Telecommunications for Space and Aeronautics, Toulouse, France
†Federation ENAC ISAE-SUPAERO ONERA, University of Toulouse, France
‡Department of Signal and Artificial Intelligence, IPSA, Toulouse, France
§Thales Alenia Space, Toulouse, France

***Abstract*—State estimation with Kalman Filters (KF) regularly encounters covariance matrices that are unknown or empirically determined, causing sub-optimal performances. Solutions to lift these uncertainties are opening up to estimation techniques based on the hybridization of KF with deep learning methods. In fact, inferring covariance matrices from neural networks gives rise to enforcing symmetric positive definite outputs. In this work, a new Recurrent Neural Network (RNN) model is explored, based on the geometric properties of the Riemannian Symmetric Positive Definite (SPD) manifold. To do so, a neuron function is defined based on the Riemannian exponential map, depending on unknown weights lying on the tangent space of the manifold. In this way, a Riemannian cost function is deduced, enabling to learn the weights as Euclidean parameters with a conventional Gauss-Newton algorithm. It involves the computation of a closed-form Jacobian. Through optimization on a simulated covariance dataset, we demonstrate the possibilities of this new approach for RNNs.**

***Index Terms*—Deep learning, RNN, Riemannian manifolds, geometry, covariance matrices.**

I. Introduction

COVARIANCE matrices and their structure have been a focal point in several problems for a wide range of fields and related applications. In image processing, it is used to model and classify data in the context of brain-computer interfaces [1, 2], medical imagery [3] and action recognition [4]. Similarly, we find applications in radar [5, 6] for signal detection. The common denominator of these methods is that they focus on the exploitation of symmetric positive definite (SPD) manifold structure.

In literature, taking advantage of the covariance structure was first issued for a context of parametric estimation. Particularly, it is possible to define an intrinsic Riemannian algorithm allowing to estimate numerically a Maximum Likelihood Estimator (MLE). It can be obtained with an estimation algorithm based on Riemannian Gaussian distribution, providing a dedicated Riemannian gradient algorithm. In [7], it is used to compute the barycenter associated to covariance matrices data following a Riemannian Gaussian distribution. Also, in the case of Euclidean observations, [8, 9] exploit the Riemannian intrinsic metrics to build a robust covariance estimator for change detection.

Abstract submitted January 12, 2024 to EUSIPCO 2024, special session. (*Corresponding author: Léa Dubreil, lea.dubreil@tesa.prd.fr*)
This work has been supported by the cooperative research laboratory in Telecommunications for Space and Aeronautics (TéSA).

In the past few decades, the embedding of machine learning techniques in statistical signal processing has risen in the context of Riemannian geometry. Enforcing the Riemannian structure of the covariance matrices in machine learning algorithm has been approached in computer vision, which has been one of the leading fields [10]. The seminal work in [4] derives a Riemannian convolutional neural network (CNN) from the SPD manifold. Other usage of SPD manifold in [1, 2] extract spatial information in electroencephalograms. Specifically in [2], they modify the kernel of a Support Vector Machine (SVM). In [1], they transform two supervised classification algorithms, Minimum-Distance to Mean (MDM) and Linear Discriminant Analysis (LDA), with the Riemannian mean and a direct classification from the tangent space.

In our current framework, the system evolves in a dynamic context, yielding time varying data, i.e. time series. In this situation, not only covariance matrices can be unknown but also other parameters controlled by a prior evolution model. The former can be modelled in deep learning by Recurrent Neural Networks (RNNs), specifically designed to deal with sequential data [11]. The latter is perfectly represented in Kalman Filters (KF), assuming Gaussianity and independence. Thus, the considered problem may involve the hybridization of RNN with a KF. In applications such as GNSS navigation, estimating the covariance matrix of the unknown parameters is often necessary because the uncertainties on the observation and propagation models are often not-well known. This issue is due to the fact that our system deals with harsh environments [12], which creates a mismatched a priori model. One axis of research is to predict these covariance matrices by a hybridized RNN-KF, accounting for the change in the observation data. This solution enables a data-driven approach of uncertainty characterization, based on observables usually not used in the empirical models predicting these covariance matrices.

In the conventional Euclidean literature, there exist two main approaches having the same ambition of embedding RNNs in KF: KalmanNet [13, 14] and DANSE [15]. Nonetheless, these methods do not verify or enforce the properties of symmetry, positive definiteness of the predicted matrices. After the prediction stage, they are reconstructed from Euclidean vectors learned by the RNN. Consequently, the covariance matrices model is a simplified case and does not fully model the true correlation of the parameters.

In this work, we propose to overcome the aforementioned problem by preserving the geometrical properties of the involved covariance matrices, in order to predict covariance matrices from temporal data in a RNN. By revising the implementation of the conventional RNN model, we substitute Euclidean memory cells with covariance matrix-like cells. Doing so requires relying on the geometric properties of the SPD manifold to preserve their structure. The contributions are twofold. First, we define a novel RNN activation function, relying on the Riemannian exponential map. Thereby, the network is designed with covariance memory cell living on a SPD Riemannian manifold, for each instant. Second, the learning optimization problem is set up by defining an intrinsic criterion built from the geodesic distance of the manifold, depending on weights belonging to a tangent space to these memory cells. In order to solve it, a closed-form Jacobian matrix is derived to optimize the network parameters, the Euclidean weights.

This paper is organized as follows: Section II briefly provides the background on Riemannian manifold properties. It is followed by a reminder of conventional RNN and the development of the proposed Riemannian RNN in section III. A numerical study is given in section IV. Finally, section V presents conclusions and future axes of work.

II. Riemannian manifold of SPD matrices

This section defines the basic concepts of manifolds (II-A) and specify them for the case of the SPD manifold (II-B).

A. *On Riemannian manifolds*

A Riemannian manifold $\mathcal{M}$ is a smooth manifold [16] endowed with a *Riemannian metric*. For each element $\boldsymbol{X}$ on the manifold $\mathcal{M}$, there is an inner product $\langle\cdot,\cdot\rangle_{\boldsymbol{X}}$ defined on the tangent space $\mathcal{T}_{\boldsymbol{X}}\mathcal{M}$ that is a bilinear symmetric positive form. As the latter is a vectorial space with dimension p, we can define a basis $\{\boldsymbol{E}_i\}_{i=1}^{p}$ of $\mathcal{T}_{\boldsymbol{X}}\mathcal{M}$. By definition, $\forall \boldsymbol{U},\boldsymbol{V} \in \mathcal{T}_{\boldsymbol{X}}\mathcal{M}$, we have $\boldsymbol{U} = \sum_{i=1}^{p} u_i\boldsymbol{E}_i$, $\boldsymbol{V} = \sum_{i=1}^{p} v_i\boldsymbol{E}_i$ and the inner product defining the *metric*

$$\langle \boldsymbol{U},\boldsymbol{V}\rangle_{\boldsymbol{X}} \triangleq \sum_{i=1}^{p}\sum_{j=1}^{p} u_i v_j g_{i,j} \tag{1}$$

where $g_{i,j}$ is the element (i,j) of the symmetric matrix $\boldsymbol{G}(\boldsymbol{X})$ [16, 17]. Specifically, we consider the Riemannian manifold to be fitted with a Rao-Fisher metric as in [18, eq. 6]. Intrinsically, the metric (1) permits to measure a length along a curve on the manifold. Let us consider a curve $\gamma(t)$ and two points on that curve such as $\boldsymbol{A} = \gamma(0)$ and $\boldsymbol{B} = \gamma(1)$. The *length of the path* between $\boldsymbol{A}$ and $\boldsymbol{B}$ is defined by [16] as

$$\mathcal{L}_{\boldsymbol{A}}^{\boldsymbol{B}}(\gamma) = \int_0^1 \langle \dot{\gamma}(t)\,,\dot{\gamma}(t)\rangle_{\gamma(t)}^{1/2}\, dt\,, \tag{2}$$

with $\gamma(t) \in \mathcal{T}_{\gamma(t)}\mathcal{M}$ the tangent vector of $\gamma(t)$. Consequently, the travelled *distance* is defined by the path which minimizes the length (2) on the *geodesic* $\gamma(t)$ (the curve that minimizes the length [16]), resulting in the distance

$$d\left(\boldsymbol{A},\boldsymbol{B}\right) = \min_{\gamma} \mathcal{L}_{\boldsymbol{A}}^{\boldsymbol{B}}(\gamma)\,. \tag{3}$$

For geodesically complete manifolds [16], there exists a geodesic $\gamma(t)$ going from $\boldsymbol{A} = \gamma(0) \in \mathcal{M}$ to $\boldsymbol{B} = \gamma(1) \in \mathcal{M}$, with the associated tangent vector $\boldsymbol{V} \in \mathcal{T}_{\boldsymbol{A}}\mathcal{M}$. In this case, the exponential map is defined by

$$\exp_{\boldsymbol{A}}\left(\boldsymbol{V}\right) = \gamma(1). \tag{4}$$

B. *Specification of the SPD manifold*

Concepts defined for SPD manifolds are recalled in Fig. 1. The Riemannian SPD manifold consists in the set of $n \times n$ symmetric positive definite matrices $\mathcal{S}^{+}(n) \triangleq \{\boldsymbol{M} \in \mathcal{M}_{n\times n}(\mathbb{R})\ ,\boldsymbol{M} = \boldsymbol{M}^{\top}\ ,\boldsymbol{x}^{\top}\boldsymbol{M}\boldsymbol{x} > 0, \forall \boldsymbol{x} \in \mathbb{R}^n \setminus \{\boldsymbol{0}\}\}$. The associated tangential space of each point on the manifold is the set of $n \times n$ symmetric real matrices [19] $\mathcal{S}(n) \triangleq \{\forall \boldsymbol{M} \in \mathcal{M}_{n\times n}(\mathbb{R}) | \boldsymbol{M} = \boldsymbol{M}^{\top}\}$. For more information on SPD manifolds, the reader can refer to [19]. Given two points $(\boldsymbol{A},\boldsymbol{B}) \in \mathcal{S}^{+}(n)$ and their respective tangential spaces $\mathcal{T}_{\boldsymbol{A}}\mathcal{S}^{+}(n) \triangleq \mathcal{S}(n)$ and $\mathcal{T}_{\boldsymbol{B}}\mathcal{S}^{+}(n) \triangleq \mathcal{S}(n)$, the distance is specified from its definition as in [19, eq. 3.10-3.12] with the matrix logarithm log

$$d(\boldsymbol{A}\ ,\boldsymbol{B}) = \|\log\left(\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{B}\boldsymbol{A}^{-\frac{1}{2}}\right)\|_F. \tag{5}$$

Similarly, the Riemannian exponential map is expressed on with the matrix exponential exp the SPD manifold

$$\begin{aligned} \boldsymbol{B} &\triangleq \exp_{\boldsymbol{A}}\left(\boldsymbol{V}\right) & (6)\\ &= \boldsymbol{A}^{\frac{1}{2}} \exp\left(\boldsymbol{A}^{-\frac{1}{2}}\boldsymbol{V}\boldsymbol{A}^{-\frac{1}{2}}\right)\boldsymbol{A}^{\frac{1}{2}}. & (7)\end{aligned}$$

In Riemannian geometry when the point $\boldsymbol{B}$ is in the neighbourhood of $\boldsymbol{A}$, there is a local diffeomorphism [16, 19] of the exponential map from the manifold to the tangential space defined by the inverse map $\log_{\boldsymbol{A}}\left(\boldsymbol{B}\right) = \exp_{\boldsymbol{A}}^{-1}\left(\boldsymbol{V}\right)$ and recalled in Fig 1.

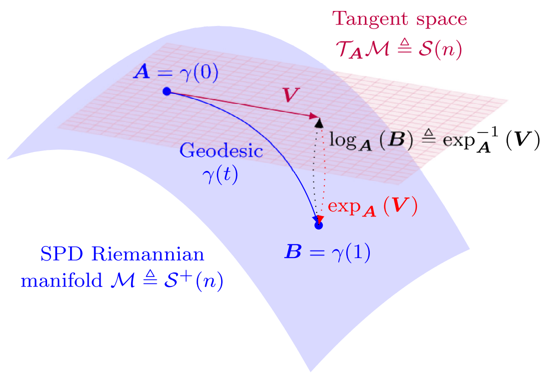
Tangent space
$\mathcal{T}_{\boldsymbol{A}}\mathcal{M} \triangleq \mathcal{S}(n)$
$\boldsymbol{A} = \gamma(0)$
$\boldsymbol{V}$
$\log_{\boldsymbol{A}}(\boldsymbol{B}) \triangleq \exp_{\boldsymbol{A}}^{-1}(\boldsymbol{V})$
Geodesic
$\gamma(t)$
$\exp_{\boldsymbol{A}}(\boldsymbol{V})$
SPD Riemannian
manifold $\mathcal{M} \triangleq \mathcal{S}^{+}(n)$
$\boldsymbol{B} = \gamma(1)$

Fig. 1. Representation of the SPD manifold $\mathcal{S}^{+}(n)$ with its associated tangent space at point $\boldsymbol{A}$ of symmetric real square matrices $\mathcal{S}(n)$. The distance between $\boldsymbol{A}$ and $\boldsymbol{B}$ is computed along the geodesic $\gamma(t)$.

III. Proposed Riemannian RNN

This section presents the proposed RNN modelling on the Riemannian SPD manifold to predict covariance matrices. First, we recall the classical Euclidean modelling of RNN in III-A. Then, we introduce the geometrical modelling by defining a new RNN activation function depending on the RNN's weight parameters lying on the tangent space of the SPD manifold in III-B. Finally, we describe the algorithmic strategy to learn the network parameters in III-C.

A. Reminder on RNN neuron and activation function

As illustrated in Fig. 2, a RNN is characterized by temporal hidden states $\{\boldsymbol{h}_t\}_{t=1}^{T} \in \mathbb{R}^d$ sequentially connecting each node cell. At each epoch, the latter is modelled by the neuron function f (8). It depends on a data input $\boldsymbol{x}_t \in \mathbb{R}^n$ and unknown weight and bias parameters $\boldsymbol{\theta}_h = \begin{bmatrix}\boldsymbol{W}_h & \boldsymbol{b}_h\end{bmatrix} \in \mathbb{R}^{n\times(n+d+1)}$. The output of each node cell is propagated in a second non-linear function g (9) which provides the RNN output $\boldsymbol{o}_t$ at each instant t, based on another set of parameters $\boldsymbol{\theta}_o = \begin{bmatrix}\boldsymbol{V}_o & \boldsymbol{b}_o\end{bmatrix} \in \mathbb{R}^{n\times(n+d+1)}$. In both equations, the activation function σ_t introduces non-linearity to the linear transformations. In literature, this function is by default an hyperbolic tangent [11]. Fig. 2 represents these equations for one instant t.

$$\boldsymbol{h}_t \triangleq f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta}_h) = \sigma_t\left(\boldsymbol{W}_h \begin{bmatrix}\boldsymbol{h}_{t-1}\\ \boldsymbol{x}_t\end{bmatrix} + \boldsymbol{b}_h\right), \quad (8)$$

$$\boldsymbol{o}_t \triangleq g(\boldsymbol{h}_t; \boldsymbol{\theta}_o) = \sigma_t\left(\boldsymbol{V}_o \boldsymbol{h}_t + \boldsymbol{b}_o\right). \quad (9)$$

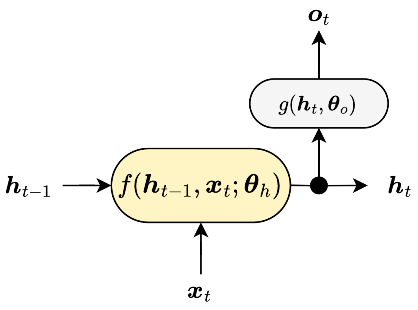
$\boldsymbol{o}_t$
$g(\boldsymbol{h}_t, \boldsymbol{\theta}_o)$
$\boldsymbol{h}_{t-1}$
$f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta}_h)$
$\boldsymbol{h}_t$
$\boldsymbol{x}_t$

Fig. 2. RNN cell for one epoch. The yellow block represents the neuron function $f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta}_t)$. The grey block is the output function $g(\boldsymbol{h}_t; \boldsymbol{\theta}_o)$.

B. Riemannian RNN model

In the following, we propose to adapt the model in Fig. 2 to change the hidden states $\boldsymbol{h}_t$ to covariance matrices $\boldsymbol{H}_t$ of dimension $n\times n$, instead of Euclidean vectors. Thus, it requires to modify the way they are handled throughout the RNN cell.

First, without considering the input vector $\boldsymbol{x}_t$, we define a model that ensures that the input/output pair belongs to the space of SPD matrices. Using the exponential map defined in (6), we can define an input-free model. Let there be two covariance matrices hidden states $\boldsymbol{\Sigma}_t \in S^+(n)$ and $\boldsymbol{H}_{t-1} \in S^+(n)$. By defining the weight matrix $\boldsymbol{W}_h$ lying on the tangent space $\mathcal{S}(n)$ at $\boldsymbol{H}_{t-1}$, the proposed generalization of the neuron function (8) on the SPD manifold yields

$$\boldsymbol{\Sigma}_t = \exp_{\boldsymbol{H}_{t-1}}(\boldsymbol{W}_h). \quad (10)$$

To take into account the contribution of the input vector $\boldsymbol{x}_t$, we integrate it by using an operation generalizing the notion of addition law in Euclidean spaces. It can be handled by the *group action* (11). More precisely, an element $\boldsymbol{Y}$ lying on the group of symmetric matrices $\mathcal{S}(n)$ can be associated to an element $\boldsymbol{Z}$ of $\mathcal{M}$. The result provides an new element in $S^+(n)$ and can be written as a function $\mathfrak{g} : \mathcal{S}(n)\times S^+(n) \to S^+(n)$ with the group action yielding

$$\mathfrak{g}(\boldsymbol{Y}, \boldsymbol{Z}) = \boldsymbol{Y}\boldsymbol{Z}\boldsymbol{Y}^\top. \quad (11)$$

First, we transform the input vector into a weighted symmetric matrix

$$\boldsymbol{X}_t = \pi^{-1}(\boldsymbol{W}_x \boldsymbol{x}_t). \quad (12)$$

In (12), the function π^{-1} is the symmetrization function, which is the inverse of the π function

$$\pi : \begin{cases}\mathcal{S}(n) & \to \quad \mathbb{R}^p \\ \boldsymbol{A} & \mapsto \quad \boldsymbol{a} = \mathrm{vech}(\boldsymbol{A})\end{cases} \quad \text{with} \quad p = \frac{n(n+1)}{2}, \quad (13)$$

with vech : $\mathcal{S}(n) \to \mathbb{R}^p$ being the half vectorization applicable to symmetric matrices. Then, incorporating (12) through the group action (11) in the input-free neuron function (6) yields the completed neuron function

$$f(\boldsymbol{H}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta}) = \boldsymbol{X}_t \boldsymbol{\Sigma}_t \boldsymbol{X}_t^\top, \quad (14)$$

which can be detailed in

$$f(\boldsymbol{H}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta}) = \pi^{-1}(\boldsymbol{W}_x \boldsymbol{x}_t)\, \boldsymbol{H}_{t-1}^{\frac{1}{2}} \exp\left(\boldsymbol{H}_{t-1}^{-\frac{1}{2}} \boldsymbol{W}_h \boldsymbol{H}_{t-1}^{-\frac{1}{2}}\right) \boldsymbol{H}_{t-1}^{\frac{1}{2}} \pi^{-1}(\boldsymbol{W}_x \boldsymbol{x}_t)^\top. \quad (15)$$

The network parameters vector is defined by $\boldsymbol{\theta} = \begin{bmatrix}\boldsymbol{W}_h & \boldsymbol{W}_x\end{bmatrix}^\top \in \mathbb{R}^{n\times(n+p)}$ with $\boldsymbol{W}_h \in S(n)$ and $\boldsymbol{W}_x \in \mathbb{R}^{p\times n}$. The overall network is thus expressed as in Fig. 3, for one instant t.

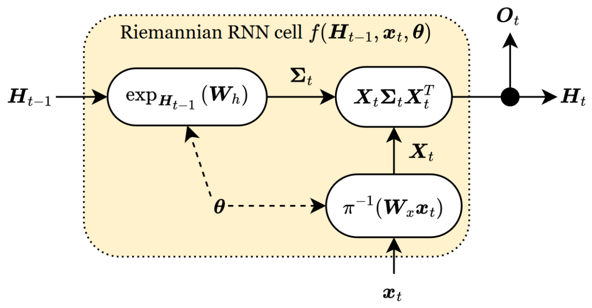
Riemannian RNN cell $f(\boldsymbol{H}_{t-1}, \boldsymbol{x}_t, \boldsymbol{\theta})$
$\boldsymbol{O}_t$
$\boldsymbol{H}_{t-1}$
$\exp_{\boldsymbol{H}_{t-1}}(\boldsymbol{W}_h)$
$\boldsymbol{\Sigma}_t$
$\boldsymbol{X}_t\boldsymbol{\Sigma}_t\boldsymbol{X}_t^T$
$\boldsymbol{H}_t$
$\boldsymbol{X}_t$
$\boldsymbol{\theta}$
$\pi^{-1}(\boldsymbol{W}_x\boldsymbol{x}_t)$
$\boldsymbol{x}_t$

Fig. 3. Riemannian RNN neuron cell for one instant t. The yellow blocks represent the network neuron function $f(\boldsymbol{H}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta})$. The network parameters $\boldsymbol{\theta}$ are used via the dashed lines in different sub-cells of the neuron function.

C. Learning problem

Let us derive the method to learn the RNN Euclidean parameters $\boldsymbol{\theta}$. Considering that the set of inputs/outputs is $\mathcal{D} = \{\boldsymbol{x}_t\ , \boldsymbol{O}_t\}\ \forall t \in [\![1, T]\!]$ as defined in Fig. 3, learning the parameters can be achieved by resolving the following geometric least-squares optimization problem

$$(\mathcal{P}) : \min_{\boldsymbol{\theta}} \sum_{t=1}^{T} d\left(\boldsymbol{O}_t, \hat{\boldsymbol{O}}_t\right)^2 \quad (16)$$

$$\Leftrightarrow (\mathcal{P}) : \min_{\boldsymbol{\theta}} \sum_{t=1}^{T} \|\log\left(\boldsymbol{O}_t^{-\frac{1}{2}} f(\boldsymbol{H}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta})\, \boldsymbol{O}_t^{-\frac{1}{2}}\right)\|_F^2. \quad (17)$$

The problem (17) is based on the geodesic distance defined in (5). For concision of writing, we consider that $g(\boldsymbol{\theta}) = \boldsymbol{O}_t^{-\frac{1}{2}} f(\boldsymbol{H}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta})\, \boldsymbol{O}_t^{-\frac{1}{2}}$ and we underline that $\boldsymbol{O}_t = \boldsymbol{H}_t$, as shown in Fig. 3. Plus, the log function being symmetric, we can reformulate the problem into a vectorization of the criterion in a Euclidean norm. It yields

$$(\mathcal{P}) : \min_{\boldsymbol{\theta}} \sum_{t=1}^{T} \|\mathrm{vec}\left(\log\left(g(\boldsymbol{\theta})\right)\right)\|^2 \quad (18)$$

$$\Leftrightarrow (\mathcal{P}) : \min_{\boldsymbol{\theta}} \sum_{t=1}^{T} \|\phi(\boldsymbol{\theta})\|^2, \quad (19)$$

with vec $: \mathcal{M}(n) \to \mathrm{R}^{n^2}$ the vectorization function of squared matrices. Equation (18) being a quadratic problem, the optimisation can be resolved using a Gauss-Newton algorithm [20]. Thus, we compute the Jacobian matrix

$$J_\phi(\boldsymbol{\theta}) = \frac{\partial \phi(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \,. \tag{20}$$

To perform this computation, we decompose $\boldsymbol{W}_h$ and $\boldsymbol{W}_x$ with the symmetric generator matrices $\boldsymbol{G}_i$ and non square generator matrices $\boldsymbol{E}_i$, respectively yielding

$$\boldsymbol{W}_h = \sum_{i=1}^{p} w_{h,i}\boldsymbol{G}_i \quad \text{and} \quad \boldsymbol{W}_x = \sum_{i=1}^{q} w_{x,i}\boldsymbol{E}_i \,, \tag{21}$$

with $q = np$ and $\boldsymbol{w}_{h,1:p} = [w_{h,1}, \ldots, w_{h,p}]$, $\boldsymbol{w}_{x,1:q} = [w_{x,1}, \ldots, w_{x,q}]$ the coefficient vectors of their weight matrices $\boldsymbol{W}_h$ and $\boldsymbol{W}_x$ after decomposition in their respective base generator. Consequently, the Jacobian matrix in (20) gives

$$J_\phi(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial \phi(\boldsymbol{\theta})}{\partial \boldsymbol{w}_{h,1:p}} & \frac{\partial \phi(\boldsymbol{\theta})}{\partial \boldsymbol{w}_{x,1:q}} \end{bmatrix} \,, \tag{22}$$

1) Derivative w.r.t. $\boldsymbol{W}_h$: To differentiate with regards to the weights applied to the memory cell, we consider the decomposition in base generator as expressed in (21). Consequently, one can write, for $l \in [\![1, p]\!]$, that

$$\frac{\partial \phi(\boldsymbol{\theta})}{\partial w_{h,l}} = \operatorname{vec}\left(\frac{\partial g(\boldsymbol{\theta})}{\partial w_{h,l}} g(\boldsymbol{\theta})^{-1}\right) \tag{23}$$

$$= \operatorname{vec}\left(\boldsymbol{H}_t^{-\frac{1}{2}} \boldsymbol{G}_l \boldsymbol{X}_t^{\boldsymbol{\theta}} \boldsymbol{\Sigma}_t \left(\boldsymbol{X}_t^{\boldsymbol{\theta}}\right)^T \boldsymbol{H}_t^{-\frac{1}{2}} g(\boldsymbol{\theta})^{-1}\right) \tag{24}$$

with $\boldsymbol{X}_t^{\boldsymbol{\theta}} \triangleq \boldsymbol{X}_t$ is defined to highlight its dependence on $\boldsymbol{\theta}$.

2) Derivative w.r.t. $\boldsymbol{W}_x$: Similarly to the derivative w.r.t. $\boldsymbol{W}_h$, we use the decomposition in (21). For $l \in [\![1, q]\!]$, the derivative is expressed as

$$\frac{\partial \phi(\boldsymbol{\theta})}{\partial w_{x,l}} = \operatorname{vec}\left(\frac{\partial g(\boldsymbol{\theta})}{\partial w_{x,l}} g(\boldsymbol{\theta})^{-1}\right) \,. \tag{25}$$

By using the expression in (15), it yields

$$\frac{\partial \phi(\boldsymbol{\theta})}{\partial w_{x,l}} = \operatorname{vec}\Big(2\boldsymbol{H}_t^{-\frac{1}{2}} \pi^{-1}(\boldsymbol{E}_l \boldsymbol{x}_t) \boldsymbol{\Sigma}_t \left(\boldsymbol{X}_t^{\boldsymbol{\theta}}\right)^T \boldsymbol{H}_t^{-\frac{1}{2}} g(\boldsymbol{\theta})^{-1}\Big) \,. \tag{26}$$

IV. Numerical experimentation

In this section, we perform a numerical study to validate the proposed RNN model defined in section III-B with the Gauss-Newton algorithm derived in section III-C. To this end, we set up two numerical tests: proof of convergence in IV-B1, robustness to the observation noise standard deviation in IV-B2.

A. Data generation

The dataset $\mathcal{D} = \{\boldsymbol{x}_t \,, \boldsymbol{O}_t\}_{t=1}^T$ simulated for our numerical validation is based on a mismatched version of the geometric model described in (27)-(28) where the dependence on the weight parameters of $\boldsymbol{x}_t$ is not explicit. In this way, we can test the proposed modelling. More precisely, we generate a noisy temporal series of covariance matrices $\{\boldsymbol{O}_t\}_{t=1}^T$ such as

$$\mathcal{D} = \begin{cases} [\boldsymbol{x}_t]_{i \in [\![1,N]\!]} & \sim \mathcal{U}(0,1) \\ \boldsymbol{O}_t & = \boldsymbol{H}_t = \boldsymbol{N}_t \boldsymbol{\Sigma}_t \boldsymbol{N}_t^\top \end{cases} \,, \tag{27}$$

$$\text{with} \begin{cases} \boldsymbol{\Sigma}_t & = \boldsymbol{H}_{t-1}^{\frac{1}{2}} \exp\left(\boldsymbol{H}_{t-1}^{\frac{-1}{2}} \boldsymbol{W}_h \boldsymbol{H}_{t-1}^{\frac{-1}{2}}\right) \boldsymbol{H}_{t-1}^{\frac{1}{2}} \\ \boldsymbol{N}_t & = \pi^{-1}(\boldsymbol{n}) \,,\ \boldsymbol{n} \sim \mathcal{N}(\boldsymbol{0}, \sigma_n^2 \boldsymbol{I}) \\ \boldsymbol{\theta} & = \begin{bmatrix} \boldsymbol{W}_h & \boldsymbol{W}_x \end{bmatrix} = \begin{bmatrix} \mathbb{1}_{n\times n} & \mathbb{1}_{p\times n} \end{bmatrix} \end{cases} \,. \tag{28}$$

The parameters of simulation are $\boldsymbol{H}_0 = \exp(\delta \boldsymbol{I}_n) \,, \delta = 0.1$, $T = 100$, and $\sigma_n = 10^{-2}$.

B. Evaluation and results

The considered simulations consist of performing a conventional Gauss-Newton algorithm to find the weights parameters $\widehat{\boldsymbol{\theta}}$ that minimizes problem (18) for one node of the Riemannian RNN. To assess the capabilities of the proposed Riemannian RNN, we first validate this performance in terms of precision, then study its robustness in terms of convergence speed of the algorithm. The latter is characterized for different values of observation noise variance.

1) Algorithm performance: For covariance sizes of $N = \{2,\ 4\}$, we study the convergence of the criterion $\phi(\boldsymbol{\theta})$ throughout the iterations of the Gauss-Newton. To achieve that, we initialize the Gauss-Newton algorithm with $\boldsymbol{W}_h^0 = (1 + err)\boldsymbol{W}_h$ and $\boldsymbol{W}_x^0 = (1 + err)\boldsymbol{W}_x$, with $err = 0.5$, being the initial error factor considered and $i = 10$, the number of Gauss-Newton iterations. We notice in Fig. 4 that in less than 3 iterations, the criterion is stable and is floored for both values of N. Convergence is fast in terms of iterations of the Gauss-Newton, demonstrating that the algorithm chosen is appropriately optimizing the cost function defined in (19).

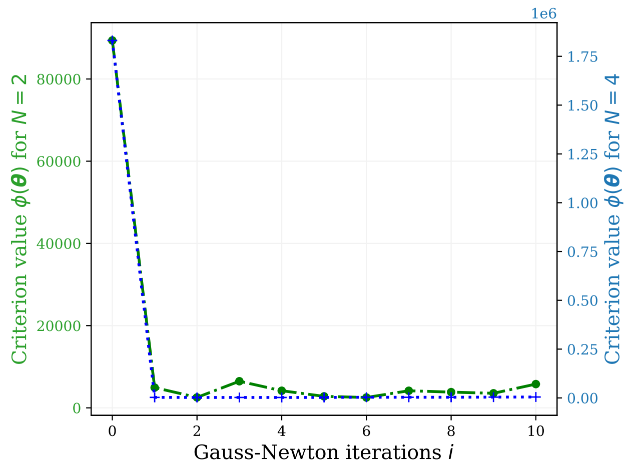

Fig. 4. Convergence test: criterion value as function of iterations i for covariance size $n = \{2, 4\}$, noise variance $\sigma_n = 10^{-2}$ and $T = 100$ samples.

2) Test of robustness: The second test studies the robustness of the optimization algorithm firstly to the observation noise standard deviation. Considering that the latter takes its values in $\sigma_n \in [10^{-3},\ 10^{-1}]$, we simulate 1000 Monte-Carlo for a covariance size of $n = \{2, 4\}$, and draw the number of the iterations necessary to converge. In Fig. 5, we observe that the number of iterations is increasing along with the observation variance. Nevertheless, it is worth noting that this increase becomes almost negligible for high standard deviation values which demonstrates a degree of robustness in terms of optimization.

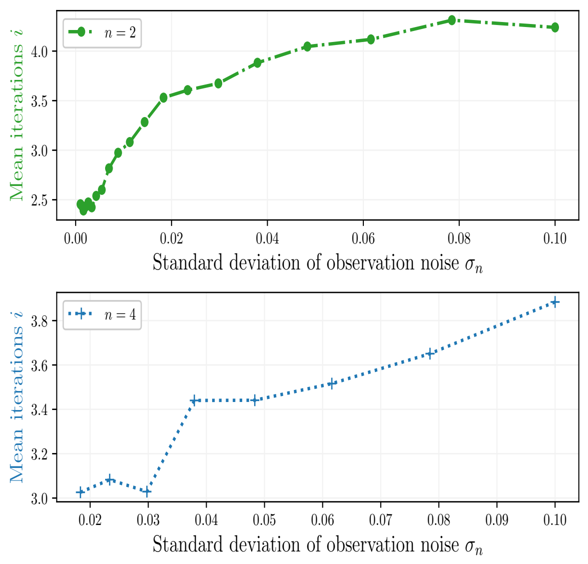

Fig. 5. Robustness test: mean Gauss-newton iterations as a function of observation noise standard deviation σ_n, for $T = 100$ observations and covariance size $n = \{2, 4\}$.

V. Conclusions and Perspectives

In this work, we proposed a novel RNN modelling, based on Riemannian geometry of SPD matrices. It allows to preserve geometrical constraints on the covariance matrices predicted by the network. The Riemannian RNN has demonstrated to be optimizable on Euclidean spaces with a standard Gauss-Newton algorithm. Having the learning problem thus formulated enables future work on a full network training, based on Euclidean optimization. They point of our method is the Euclidean structure of the unknown parameters. Thus, it can be learned using a conventional gradient algorithm without the need for Riemannian optimization. Besides, the model has proven to be robust to certain levels of noise ($\sigma_n < 10^{-1}$). Having a non convex and quadratic criterion, the execution time of the optimization increases along with the covariance size n. In fact, these Riemannian-based methods are subject to computational complexity and numerical instability due to the bad conditioning of high dimensions SPD matrices [21]. Perspectives of improvement can be inspired from robust dimension reduction for SPD matrices, which has been researched in [22, 23]. This could lead to an enhancement of the aforementioned model. Future works on the model will focus on two axes: (1) consider the output equation (9) to be refined in Riemannian way; (2) estimate the neuron function parameters following an unsupervised paradigm to be consistent with an unsupervised dynamic framework in the context of GNSS navigation.

References

[1] Alexandre Barachant et al. “Multiclass Brain–Computer Interface Classification by Riemannian Geometry”. In: *IEEE Transactions on Biomedical Engineering* 59.4 (2012), pp. 920–928.

[2] Alexandre Barachant et al. “Classification of covariance matrices using a Riemannian-based kernel for BCI applications”. In: *Neurocomputing* 112 (2013), pp. 172–178.

[3] Maria Sayu Yamamoto et al. “Novel SPD Matrix Representations Considering Cross-Frequency Coupling for EEG Classification Using Riemannian Geometry”. In: *2023 31st European Signal Processing Conference (EUSIPCO)*. 2023, pp. 960–964.

[4] Zhiwu Huang and Luc Van Gool. “A Riemannian Network for SPD Matrix Learning”. In: *Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*. 2017, pp. 2236–2242.

[5] Marc Arnaudon, Frédéric Barbaresco, and Le Yang. “Riemannian Medians and Means With Applications to Radar Signal Processing”. In: *IEEE Journal of Selected Topics in Signal Processing* 7.4 (Aug. 2013), pp. 595–604.

[6] Xiaoqiang Hua et al. “Log-Euclidean Metric-Based Signal Detector with Manifold Filter and Matrix Information Geometry”. In: *2020 IEEE 20th International Conference on Communication Technology (ICCT)*. 2020, pp. 1208–1212.

[7] Paolo Zanini et al. “Parameters estimate of Riemannian Gaussian distribution in the manifold of covariance matrices”. In: *2016 IEEE Sensor Array and Multichannel Signal Processing Workshop (SAM)*. 2016, pp. 1–5.

[8] Florent Bouchard et al. “Riemannian Framework for Robust Covariance Matrix Estimation in Spiked Models”. In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 5979–5983.

[9] Florent Bouchard et al. “Riemannian geometry for compound Gaussian distributions: Application to recursive change detection”. In: *Signal Processing* 176 (Nov. 2020), p. 107716.

[10] Pavan Turaga and Anuj Srivastava. *Riemannian Computing in Computer Vision*. Jan. 2016.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. ISBN: 978-0-262-33737-3.

[12] Antonio Angrisano, Salvatore Gaglione, and Antonio Maratea. “A comparison between resistant GNSS positioning techniques in harsh environment”. In: *2018 European Navigation Conference (ENC)*. 2018, pp. 140–147.

[13] Guy Revach et al. “Unsupervised Learned Kalman Filtering”. In: *2022 30th European Signal Processing Conference (EUSIPCO)*. 2022, pp. 1571–1576.

[14] Guy Revach et al. “KalmanNet: Neural Network Aided Kalman Filtering for Partially Known Dynamics”. In: *IEEE Transactions on Signal Processing* 70 (2022), pp. 1532–1547.

[15] Anubhab Ghosh, Antoine Honoré, and Saikat Chatterjee. “DANSE: Data-driven Non-linear State Estimation of Model-free Process in Unsupervised Learning Setup”. In: *2023 31st European Signal Processing Conference (EUSIPCO)*. 2023, pp. 870–874.

[16] Xavier Pennec. “Intrinsic Statistics on Riemannian Manifolds: Basic Tools for Geometric Measurements”. In: *Journal of Mathematical Imaging and Vision* 25.1 (2006), pp. 127–154.

[17] Manfredo Perdigão do Carmo. *Riemannian Geometry*. Birkhäuser Boston, 1992.

[18] Salem Said et al. “Riemannian Gaussian Distributions on the Space of Symmetric Positive Definite Matrices”. In: *IEEE Transactions on Information Theory* 63.4 (2017).

[19] Alexander Smith et al. “Data Analysis using Riemannian Geometry and Applications to Chemical Engineering”. In: *Computers and Chemical Engineering* 168 (2022).

[20] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. ISBN: 978-0-387-30303-1.

[21] Rajendra Bhatia. *Positive Definite Matrices*. Princeton Series in Applied Mathematics. Princeton (N.J.): Princeton University Press, 2007. ISBN: 978-0-691-12918-1.

[22] M. Congedo et al. “A closed-form unsupervised geometry-aware dimensionality reduction method in the Riemannian Manifold of SPD matrices”. In: *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2017, pp. 3198–3201.

[23] Alireza Davoudi, Saeed Shiry Ghidary, and Khadijeh Sadatnejad. “Dimensionality reduction based on Distance Preservation to Local Mean (DPLM) for SPD matrices and its application in BCI”. In: *Journal of Neural Engineering* 14.3 (2017).