UCLouvain

FACULTE DES SCIENCES

Faculté des sciences

Wavelet thresholding for symmetric positive-definite matrix-valued curves in a log-Euclidean manifold

Auteur : Yannic WACHEL
Promoteur : Pr. Rainer VON SACHS
Lecteurs : Pr. Rainer VON SACHS, Pr. Christian HAFNER
Année académique 2021-2022
Master en statistique, orientation générale, à finalité approfondie

Wavelet thresholding for symmetric positive-definite matrix-valued curves in a log-Euclidean manifold

Wachel Yannic

August 2022

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Pr. Rainer von Sachs, for his kindness, comprehensiveness and for his supervision during my thesis' journey. Not only you have been there to guide me and give me your input on the statistical aspects of this master thesis, but you have also showed a great patience given my personal and professional situation. I can't thank you enough for all of that.

Also, I would like to thank my family, especially the love of my life, Alexandra, for her help and understanding throughout those years. Thank you also for our two (most of the time) amazing kids: Louis and Charles.

Abstract

In this work, we perform nonparametric estimation of symmetric and positive-definite (SPD) matrix-valued curves living in a Riemannian manifold equipped with the log-Euclidean metric. Using wavelet thresholding methods, together with a simple signal-plus-noise data model, we find several thresholding rules in the wavelet domain, such as a generic non-scalar signal-plus-noise expression. We implement a sum-thresholding (ST) rule that we compare with an existing trace-thresholding (TT) rule, derived in the context of Hermitian positive-definite matrices in the affine-invariant manifold. The estimator found with the ST rule using the log-Euclidean metric outperforms in terms of time computation the estimator found with the TT rule using the affine-invariant (and log-Euclidean) metrics. Finally, we find that equivariance of an estimator under orthogonal congruence transformations is a general feature shared within both rules with respect to their related metrics.

Contents

Notation and symbols, abbreviations

The next list describes several notation and symbols, together with the abbreviations used within the body of the document.

Symbols and notations

$\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{C}$	Sets of natural, integer, real and complex numbers
$\mathcal{M}_{(\cdot)}$	Riemannian manifold, the subscript refers to the metric used
$\cong$	Vector space isomorphism
$Sym^{++}(n)$	Set of symmetric and positive-definite $n \times n$ matrices
$\odot$	Product of matrices defined on $Sym^{++}(n)$
$\otimes$	Product by a real number defined on $Sym^{++}(n)$
δ_R, δ_{LE}	Distance induced by Riemannian metric, either affine-invariant (subscript R) or log-Euclidean (subscript LE)
$\sim$	Distributed as or proportional to
$\delta_{i,j}$	Kronecker delta
$\eta(p_1, p_2, t)$	Geodesic segment connecting p_1 and p_2 with $0 \leq t \leq 1$
δ_w	Dirac delta, centered on w
$\langle \cdot, \cdot \rangle_{(\cdot)}$	Frobenius or log-Euclidean inner product, the subscript refers to the metric used
$\|\| \cdot \|\|_{(\cdot)}$	Frobenius or log-Euclidean norm, the subscript refers to the metric used
$L_p(X)$	L_p-space of p-integrable functions on X (w.r.t. Lebesgue measure)
$T_p(\mathcal{M})$	Tangent space to $\mathcal{M}$ at p
$\mathrm{GL}(d, \mathbb{K})$	General linear group of $(d \times d)$-dimensional invertible complex (if $\mathbb{K} = \mathbb{C}$) or real ($\mathbb{K} = \mathbb{R}$) matrices
g, g_R, g_{LE}	Riemannian metric, the subscript refers to the metric used
$\mathrm{Exp}_p(v)$	Exponential map at p in direction of v
$\mathrm{Log}_p(v)$	Logarithmic map of q at p
$\log(\cdot)$	Matrix logarithm
$\exp(\cdot)$	Matrix exponential

$\mathrm{Tr}(\cdot)$	Trace of a matrix
y^*	Conjugate transpose of a matrix y
$y * x$	Matrix congruence transformation y^*xy
y^t, y^T	Transpose of a matrix y
$\mathbb{E}(\cdot), E(\cdot)$	Intrinsic (Karcher or Fréchet) mean on a Riemannian manifold, Euclidean expected value operator
$P(\mathcal{M})$	Set of probability measures on a Riemannian manifold
$P_p(\mathcal{M})$	Set of probabilities in $P(\mathcal{M})$ with finite moments of order p
μ	Intrinsic (Karcher) mean
$\mathrm{Ave}(\cdot, \cdot)$	Intrinsic weighted or unweighted sample mean
$\mathrm{Var}(\cdot)$	Variance of a scalar or variance-covariance of a matrix
$\mathbf{1}(\cdot)$	Matrix having 1 as components
Q, Λ	Orhtogonal matrix of eigenvectors, diagonal matrix of eigenvalues
$\mathrm{Ave}_I(\cdot)$	Intrinsic mean of manifold curve over set I
$M_{j,k}, \tilde{M}_{j,k}$	Midpoint and predicted midpoint at scale-location (j, k)
$D_{j,k}$	Wavelet coefficient at scale-location (j, k)
$\mathcal{D}_{j,k}$	Whitened wavelet coefficient at scale-location (j, k)
C_N	Filter weights in average-interpolation subdivision
$\det(\cdot)$	Determinant of a matrix
n	Usually the number of sample points or a dimension
J	Maximum wavelet scale
N	Subdivision order
σ	Standard deviation, usually related to the data model
α	Fine-tuning parameter for nonlinearity
λ	Usually wavelet (threshold) parameter
γ_t	Curve on a Riemannian manifold (ie. SPD)
d	Usually a dimension

$\mathbb{1}(\cdot)$	Indicator function with condition in $(\cdot)$

Abbreviations

SPD	Symmetric and positive-definite
HPD	Hermitian and positive-definite
PD	Positive-definite
DTI	Diffusion tensor imaging
MRA	Multiresolution analysis
MSE	Mean squared error (depends on the metric)
TT	Trace-thresholding rule
ST	Sum-thresholding rule

1 General Introduction

Classic statistical analyses of multivariate data usually involves studying the dispersion of variables or the way in which variables change in relation to one another via the so-called *variance-covariance* matrix. Estimating those matrices obviously is important in many inference problems such as the construction of confidence regions for unknown parameters, hypothesis testing, and so on. In the last decades, more interest has been drawn to covariance-type of data due to its presence in specific fields like statistical time series analysis or biology, through the study of the diffusion tensor imaging technique. More generally, they are widely used in computeur vision, for image analysis or motion analysis (Quang and Vittorio 2018), in mechanics for the study of stress tensors or in the solving of partial differential equations (PDEs) in three dimensions (Mohammadi, Borouchaki, and George 1997). Within a time series, observations are ordered and this ordering introduces a serial correlation structure between adjacent observations. In this context, the covariance matrix characterizes the linear second order dependence between the elements of the time series. Thus, in the case of vector-based time series, we end up with a collection of covariances matrix having a dependence in time or frequency which are needed to study the fluctuating or oscillating behaviour of the time series. In diffusion tensor imaging (DTI)(Bihan 1991), we are interested in mapping the structure of the brain non-invasively which is done by looking at the local movement of water molecules within a small region of the brain. This information is then summarized by a 3×3 matrix sharing similar properties as the covariance matrix. In both cases, we find ourselves with an ensemble (a *collection*) of matrices, variably changing through time or space, that we wish to use to perform statistical analyses. The shared characteristic of those matrices[1] is that they are *symmetric* and *positive definite* matrices or, in short, SPD. Formally, one can define a SPD matrix in the following way. Let Σ be a SPD matrix, for example a sample covariance matrix $\mathrm{Cov}(X_i, X_j) \in \mathbb{R}^{n \times n}$ related to a (sample) random vector living in $\mathbb{R}^n$. We have the basic following properties for Σ:

- it is symmetric: $\Sigma = \Sigma^T$
- positive semi-definite: $u^T \Sigma u \geq 0$ for any vector $u \in \mathbb{R}_0^n$

This implies that, in the spectral representation of the matrix Σ, his eigenvalues are real and strictly positive. Those SPD matrices will be our focus data, with the idea to perform statistical analysis, such as eg. nonparametric estimation. A given collection of SPD matrices, that can be seen as *curve* from a continuous standpoint, can be analysed and one might have interest in looking for a mean SPD matrix (or other related statistics) or perform operations used in classical analyses such as regularisation, interpolation, denoising and so forth. In this work, we mainly consider the aspect of denoising a given curve (or signal) of SPD matrices via nonparametric methods of estimation. Thus, we consider an observed signal $\tilde{\mathcal{S}}$, which is a curve of noisy SPD matrices, and perform nonparametric methods in order to remove the noise and keep the relevant information contained in the original signal $\mathcal{S}$. Many

[1] Covariance matrices are actually symmetric and positive (semi)-definite, but we will focus on the latter case.

nonparametric estimation methods exist and have been used to treat covariance-type of data, eg. kernel smoothing, splines, wavelet smoothing, bootstrap to name the main ones. An important aspect is that SPD matrices actually live in a specific geometry, ie. the cone of SPD matrices, which presents a curvature, unlike the Euclidean spaces. A classical way to perform estimation is to consider a curve of data points (such as SPD matrices) that is embedded in an Euclidean space of higher dimension. Doing so, we can't be sure that the statistics, such as the mean, or the estimator found will keep the SPD property. With a few exceptions like the use inflexible kernels, multitaper smoothing or Cholesky method, the nonparametric estimator is not necessarily SPD. The Euclidean embedding also has other disadvantages, such as being an incomplete metric space, ie. there isn't always a connection between two points on the cone of SPD matrices in this space. On a computational point of view, the Euclidean standpoint also generates a swelling effect, where the interpolation between SPD matrices creates an increase in dispersion which leads to a larger "volume" of the mean compared to the original data. This force us to take another approach by considering the *intrinsic* point of view of the geometry, in which our data live in a free-standing way which, theoretically, gives more sense and more clarity than the latter approach. A natural framework for our SPD matrix-valued data is to use Riemannian manifolds with the so-called affine-invariant metric (Pennec, Fillard, and Ayache 2006, Pennec 2006). This non-Euclidean approach introduces the notion of manifold or curved spaces for which basic definition and tools of statistical analysis (eg. inference and estimation problems) are more complicated, both on mathematical standpoint and in terms of computation cost and implementation.
In (Chau 2018)[2], they studied nonparametric estimation methods, using wavelets, for curves of Hermitian positive-definite (HPD) matrices living in a Riemannian manifold with the affine-invariant metric. This extensive work proposes the use of a specific thresholding rule in the wavelet domain, in order to keep the signal-related wavelet coefficients and disregarding the noise-related coefficients. This method showed a better accuracy, for the found estimator, using the affine-invariant metric than with other used metrics, such as the log-Euclidean or the Cholesky metrics, and also compared to other nonparametric estimation methods. They developed an intrinsic version of a wavelet transforms, using average-interpolation (Donoho 1993), and a nonlinear tree-structured wavelet thresholding rule to find an estimator that keeps the affine-invariant property, while staying in the space of HPD matrices. Despite showing good performances, this approach make use of complicated mathematics where the computation of statistics or measures on the manifold is quite time-consuming. This is due to the curvature induced by the affine-invariant metric. Several options have been proposed to simplify this framework, for example in (Arsigny et al. 2006, Dryden, Koloydenko, and Zhou 2009, Yuan et al. 2012, Huang 2015), where they use a new family of metrics, called log-Euclidean metrics, which allow for faster computation by the use of classical Euclidean calculus in the domain of matrix logarithms. This metric also benefits of a high degree of symmetry, avoid the aforementioned swelling problem and lead to a complete metric space.

The main goal of the present work is to consider the space of SPD matrices as a Riemannian

[2]When using this citation, we also refer to (Chau and Sachs 2016) and (Chau and Sachs 2021).

manifold with the log-Euclidean metric and perform nonparametric curve estimation using wavelet thresholding methods. With this metric, we transform the space of SPD matrices in a complete metric space and we keep the unitary congruence invariant property but not the general linear congruence invariance, that is found with the affine-invariant metric. The idea is to adapt and compare our approach to the general framework developed in (Chau 2018). In doing so, we also look for other possible data models, as an extension of our first model, using the spectral properties of SPD matrices. The structure of the this thesis is the following:

- The first part of **chapter 2** gives a (brief) introduction on non-Euclidean data, the main properties of the log-Euclidean manifold with a comparison to the affine-invariant one, and describe how probabilities are expressed in this context. The second part focus on the description of wavelets, the lifting scheme in the average-interpolation subdivision and the (non)linear wavelet thresholding methods.

- **Chapter 3** focus on building our simple signal-plus-noise model, together with the probability distribution of the noise, and looking at a possible other data model by considering the spectral representation of SPD matrices in terms of their eigenvectors and associated eigenvalues.

- In **chapter 4**, we describe how the average-interpolation is applied, with the forward/backward wavelet transform in the context of the log-Euclidean metric and we discuss the equivariance properties of both the log-Euclidean and affine-invariant metrics.

- **Chapter 5** is where we derive the link between the signal and noise in the wavelet domain, via the whitened wavelet coefficients, which allows to describe the possible thresholding rules we can apply to perform noise-removal.

– Comparisons, based on simple simulated data, are done in **chapter 6**, where we compare the trace-thresholding rule (with both the log-Euclidean and affine-invariant metrics) with the sum-thresholding rule. Average mean squared error and computation time are calculated as comparisons criteria.

2 Theoretical Background

2.1 Data living in non-Euclidean manifolds

Euclidean spaces are the usual background for classical statistics, because an Euclidean setting is the most common case and the geometry with the related mathematics (especially calculus) are well-understood. As a basic example, we can describe the study of of a linear regression between a one-dimensional variable Y in function of another (one-dimensional) variable X, which is made through observations (x_i, y_i) represented as points in a graph. Those points are in the $\mathbb{R}^2$ plane and, in case we draw a regression curve through the data, the curve itself is also in the plane. We say that the points or the curve is embedded in the space $\mathbb{R}^2$, which is a Euclidean or flat space. The regression curve is described with the use of a basis of the place, which can be actually changed into another basis. The properties (or description) of this curve that depends on the coordinate space that curve is embedded in are called *extrinsic* properties of the curve. For example, the slope of a tangent line is an extrinsic property since it depends on the coordinate system. In contrast, the *intrinsic* properties of the curve are properties that can be measured within the surface itself without any reference to a larger space. For example, the length of a curve. Those intrinsic properties are computed via the use of the *metric tensor* which characterizes the specific geometry (here, a curve) we consider.
Statistical data analysis on spheres (Rossmanith 2013), image or motion analysis (Quang and Vittorio 2018), diffusion tensor imaging (Dryden, Koloydenko, and Zhou 2009, Zhu et al. 2009), to name a few areas of research (Moakher and Batchelor 2006), are all using data that live on non-Euclidean geometries, meaning that the statistical tools, such as nonparametric estimation via wavelets, need to output objects that *still* live in the same geometry. As an example, let's consider data living on the surface of a sphere. The interpolation between two points on the sphere could be represented by the following:

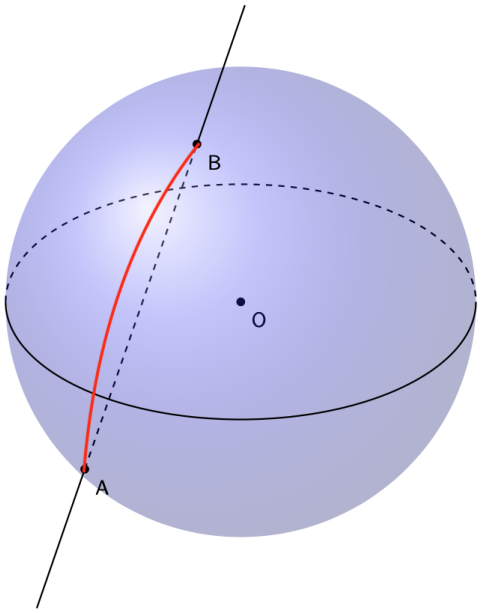


Figure 1: Interpolation on the sphere: straight black line is interpolation using the embedding Euclidean space, curved red line is interpolation using the intrinsic geometry of the sphere

Clearly, we see that the straight line, obtained from the Euclidean interpolation, goes inside the sphere which is not in the geometry because we only consider the surface of the sphere. This is not a problem with the red curve, where each points of the curve belong in the geometry (the surface of the sphere). We therefore need to implement intrinsic statistical tools, valid on the given geometry, to perform valid estimation procedures of a given signal. Contrary to the Euclidean spaces, which exhibits flatness (see for example the *plane* $\mathbb{R}^2$), a non-Euclidean geometry like the surface of the sphere in the previous example exhibits a clear *curved* geometry.

The intrinsic geometries of the HPD matrices and, their subset, the SPD matrices are both curved in nature and this curvature brings along more complexity through a different mathematical formulation. Those non-Euclidean geometries will be referred as manifolds and the mathematics on those geometries belong to the world of *differential geometry* (Boothby 1986, Lee 2013, Carmo 1992). A *manifold* is an abstract metric space that looks locally, but not necessarily globally[3], like an Euclidean space. The intrinsic measurements or calculations on or from the manifold are made via the *tensor metric*, which is a bilinear, symmetric and non-degenerate function of the manifold. In this work, we will consider smooth manifolds and, more precisely, the class of Riemannian because the latter can always be equipped with a Riemannian metric. Thus, we will write a Riemannian manifold as $(\mathcal{M}, g)$ where $\mathcal{M}$ is the manifold and g is the metric induced on the manifold. The metric is a positive-definite inner product on the tangent space $T_p(\mathcal{M})$ at each point p, and allows to define several geometric notions

[3]A good example is the space-time we are living in: it is globally non-Euclidean, eventhough the curvature seems to be very small, while being locally flat.

such as angle at an intersection, the length of a curve,the area of a surface and higher-dimensional analogues (volume, etc.).
In (Chau 2018), they performed nonparametric spectral matrix estimation in the space of HPD matrices as a Riemannian manifold equipped with a Riemannian metric, the affine-invariant metric. We will perform nonparametric estimation in the space of SPD matrices as a Riemannian manifold equipped with a log-Euclidean metric. Those manifolds, together with their structure and tools, are briefly introduced in the next section.

2.2 Symmetric Positive Definite matrices in Riemannian manifolds

For data such as HPD matrices (or a subclass, eg. SPD matrices), we actually deal with data lying on a Riemannian manifold. Our interest lies in the set of $n \times n$ SPD matrices, denoted $Sym^{++}(n)$, which, depending on the standpoint can be seen as a subset of the Euclidean space $\mathbb{R}^{n \times n}$, thus inheriting the Euclidian distance, or as a smooth manifold with either the affine-invariant or log-Euclidean distance, coming from the corresponding metrics (Bhatia 2006). With the Euclidean distance, we don't have a one-to-one correspondance between two points in $Sym^{++}(n)$, which is referred as an incomplete metric space, and eventhough we get rid of mathematical complications we still have an undesirable swelling effect: when computing the mean of a set of SPD matrices, the determinant of the Euclidean matrix mean can be strictly larger than the determinants of the original matrices. This is a big problem, given the over-dispersion induced by the swelling, especially in DTI analysis (Arsigny et al. 2006). Another issue is the fact that statistics or estimation performed on SPD matrices must still have the SPD property. Indeed, the following picture gives another reason to discard the Euclidean distance (or metric).

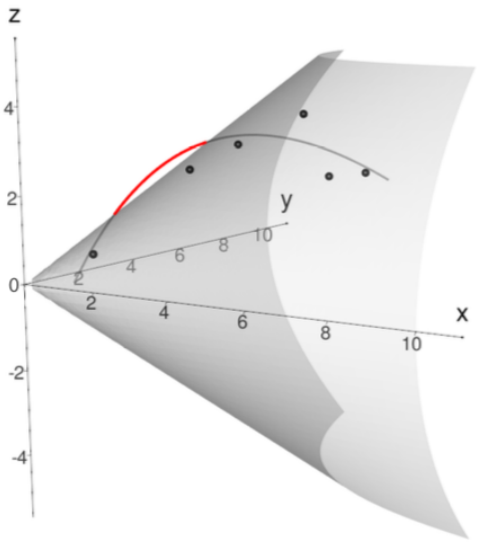
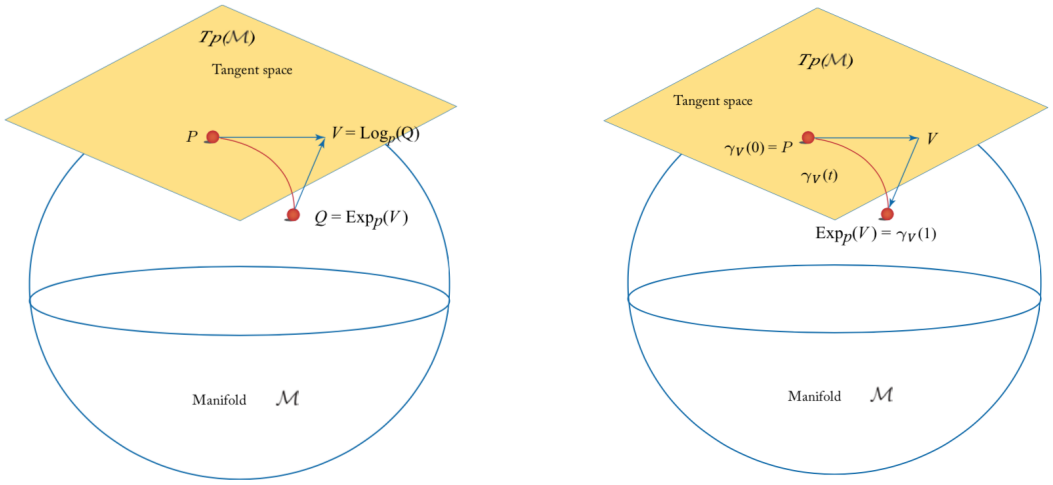


Figure 2: Euclidean polynomial regression between a set of SPD matrices. The results, in red, clearly leaves the cone of SPD matrices. (Chau 2018)

In the Riemannian representation, we take into account the intrinsic geometry of $Sym^{++}(n)$ at cost of more mathematical complexity. Below, we briefly describe some of the key concepts from Riemannian geometry:

- a Riemannian metric induces a Riemannian distance, ie. the shortest distance between two points on the manifold
- a geodesic between two points on the manifold is actually a generalization of a straight line in a Euclidean space
- the exponential map $\text{Exp}_P(V)$ makes the link between a given point P on the manifold $\mathcal{M}$ and a given tangent vector $V \in T_P(\mathcal{M})$, belonging in the tangent space. Under the Euclidean metric, $T_P(\mathcal{M} = \mathbb{R}^n) \cong \mathbb{R}^n$ and $\text{Exp}_P(V) = P + V$ where $P, V \in \mathbb{R}^n$, which corresponds to a simple addition.
- the logarithmic map is the inverse of the exponential map where, under the Euclidean metric, $T_P(\mathcal{M} = \mathbb{R}^n) \cong \mathbb{R}^n$ and $\text{Log}_P(Q) = Q - V$ where $P, V \in \mathbb{R}^n$, which corresponds to a simple soustraction.
- in a geodesically complete manifold $\mathcal{M}$, there exists a geodesic connecting any two points, whose length is equal to their Riemannian distance, this distance being the shortest.



(a) Exponential map

(b) Logarithmic map

Figure 3: Geometric representation of the Riemannian exponential and logarithmic maps (Quang and Vittorio 2018)

We are interested in the so-called Cartan-Hadamard manifolds, where the geodesic connecting two points is *unique* and its length, which is equal to their Riemannian distance, is completely determined by the Riemannian logarithm map. Let's now turn to our manifold associated with the log-Euclidean metric.

With $\mathcal{M} = Sym^{++}(n)$, we have, for each $P \in Sym^{++}(n)$, the following isomoprhism between the tangent space at P and the vector space of symmetric $n \times n$ matrices $Sym(n)$:

$$T_P(Sym^{++}(n)) \cong Sym(n)$$

Instead of using the affine-invariant metric, we want to define the log-Euclidean metric where the log-Euclidean distance will be the geodesic distance on $Sym^{++}(n)$. This metric is found by defining the following operations on $Sym^{++}(n)$:

- $\odot : Sym^{++}(n) \times Sym^{++}(n) \to Sym^{++}(n) : A \odot B = \exp(\log A + \log B)$
- $\otimes : \mathbb{R} \times Sym^{++}(n) \to Sym^{++}(n) : \lambda \otimes A = \exp(\lambda \log A) = A^{\lambda},\ \lambda \in \mathbb{R}$
- log-Euclidean inner product $\langle A, B \rangle_{LE} = \langle \log A, \log B \rangle_F = \mathrm{Tr}(\log A \log B)$

where the first two give $Sym^{++}(n)$ a vector space structure and with the last one, the log-Euclidean metric is defined. We notice that the classical exponential $\exp(\cdot)$ (logarithmic $\log(\cdot)$) of a symmetric (SPD) matrix will lead to a SPD (symmetric) matrix. An important feature to notice is that the classical matrix logarithm $\log(\cdot)$ actually leads to the following isomorphism between inner product spaces:

$$\log : (Sym^{++}(n), \odot, \otimes, \langle \cdot, \cdot \rangle_{LE}) \to (Sym(n), +, \cdot, \langle \cdot, \cdot \rangle_F) : A \to \log A \tag{1}$$

where $(+, \cdot)$ are the standard matrix addition and scalar multiplication operations, respectively. We can say that the log-Euclidean metric essentially flattens $Sym^{++}(n)$ via the previous map in the sense that it leads to classical Euclidean computations such as the use of the Frobenius norm. This will have a great impact on our calculation when considering the wavelet transforms and the obtained wavelet coefficients. Thus, we can flatten the Riemannian geometry by using the log-Euclidean metric in the Riemannian manifold having the consequence that we can use the classical Euclidean tools in the treatment of SPD matrix-valued data. Below, in order to not surcharge the section, we provide in table 1 the mathematical description of the tools used in this thesis for the affine-invariant metric and the log-Euclidean metric. The principal properties, along with some symmetries, of the relevant distances used in $Sym^{++}(n)$ are also given.

	$(\mathcal{M}_R, g_R)$ - Riemannian manifold with affine-invariant metric	$(\mathcal{M}_{LE}, g_{LE})$ - Riemannian manifold with log-Euclidean metric
Manifold	$\mathbb{P}_{n\times n} := \{p \in \mathbb{C}_{n\times n} : p = p^* \text{ and } \vec{z}^* p \vec{z} > 0, \text{ for } \vec{z} \in \mathbb{C}_n,\ \vec{z} \neq 0\}$	$Sym^{++}(n) := \{p \in \mathbb{R}_{n\times n} : p = p^t \text{ and } \vec{z}^t p \vec{z} > 0, \text{ for } \vec{z} \in \mathbb{R}_n,\ \vec{z} \neq 0\}$
Tangent spaces	$T_p(\mathbb{P}_{n\times n}) \cong \mathbb{H}_{n\times n} := \{h \in \mathbb{C}_{n\times n} : h = h^*\}$	$T_p(Sym^{++}(n)) \cong Sym(n)$
Metric	$\langle h_1, h_2\rangle_p = \mathrm{Tr}((p^{-1/2} * h_1)(p^{-1/2} * h_2))$	$\langle h_1, h_2\rangle_p = \mathrm{Tr}(d_p \log(h_1) d_p \log(h_2))$
Distance	$\delta_R(p_1, p_2) = \|\|\mathrm{Log}(p_1^{-1/2} * p_2)\|\|_F$	$\delta_{LE}(p_1, p_2) = \|\| \log(p_2) - \log(p_1)\|\|_F$
Geodesics	$\eta(p_1, p_2, t) = p_1^{1/2} * (p_1^{-1/2} * p_2)^t,\ 0 \leq t \leq 1$	$\exp((1-t)\log(p_1) + t\log(p_2))$
Exponential map	$\mathrm{Exp}_p(h) = p^{1/2} * \mathrm{Exp}(p^{-1/2} * h)$	$\mathrm{Exp}_p(h) = \exp(\log h + d_p \log h)$
Logarithmic map	$\mathrm{Log}_p(q) = p^{1/2} * \mathrm{Log}(p^{-1/2} * q)$	$\mathrm{Log}_p(q) = d_{\log(p)} \exp(\log q - \log p)$
Parallel transport	$\Gamma_q^p(w) = p^{1/2} * (p^{-1/2} * q)^{1/2} * p^{-1/2} * w$	$\Gamma_q^p(w) = d_{\log(p)} \exp(d_q \log w)$

Table 1: Summary of basic tools with $a * x = a^* x a$ being a matrix congruent transformation.

In this table, we note the presence of the so-called Fréchet derivative of the function $\log : Sym^{++}(n) \to Sym(n)$, noted $d_p \log$, which can be computed using a discrete algorithm, for eg. (Boumal and Absil 2011). In the present work, this derivative will not be necessary because of our simplified framework.

	Euclidean	log-Euclidean	affine-invariant
Geodesic distance	Yes	Yes	Yes
Affine invariance	No	No	Yes
Scale invariance	No	Yes	Yes
Unitary invariance	Yes	Yes	Yes
Inversion invariance	No	Yes	Yes
Inner product distance	Yes	Yes	No
Complete metric	No	Yes	Yes

Table 2: Properties and symmetries of the distances in the Euclidean, log-Euclidean and affine-invariant frameworks.

An interesting feature concerns a potential relation between the affine-invariant metric and the

log-Euclidean metric. The induced distances from the affine-invariant and log-Euclidean metrics can indeed be linked via the Campbell-Baker-Hausdorff formula (Hall 2003), which states that for any two matrices $A, B \in Sym^{++}(n)$, sufficiently close to the zero matrix (ie. the identity), we have

$$|| \log(A^{-1/2} B A^{-1/2})||_F^2 = || \log(A) - \log(B)||_F^2 + \frac{1}{6} \mathrm{Tr}\left([\log A, [\log A, \log B]](\log A - \log B)\right) + ...$$

which is only true close to the identity. Without being to general, we will have a look into this when comparing the affine-invariant and log-Euclidean approaches for denoising in the wavelet domain.

Given the statistical nature of the present work, we need to link the geometric framework of the log-Euclidean manifold to statistical/probabilistic aspects and the tools related. We need to define the notions of probability distributions and random variables (Arsigny et al. 2006, Chau 2018).
Let the random variable $X : (\Omega, \mathcal{A}, \nu) \to (Sym^{++}(n), \mathcal{B}(Sym^{++}(n)))$ be a measurable function where $\mathcal{B}(Sym^{++}(n))$ is the related Borel algebra and ν is the induced probability on $Sym^{++}(n)$. We will work with a subset $P_p(Sym^{++}(n))$, of all probability measures $P(Sym^{++}(n))$, that have finite moment of order p with respect to the log-Euclidean distance, δ_{LE}, that is,

$$P_p(Sym^{++}(n)) = \left\{ \nu \in P(Sym^{++}(n)) : \exists y_0 \in Sym^{++}(n) \middle| \int_{Sym^{++}(n)} \delta^p_{LE}(y_0, x)\nu(dx) < \infty \right\}$$

With those notions, we can calculate the Karcher or Fréchet mean which describes an (intrinsic) center of the random variable X with the probability measure ν. The set of Karcher means is obtained via the minimization of the second moment with respect to the log-Euclidean distance:

$$\mathbb{E}(X) = \text{argmin}_{y \in supp(\nu)} \int_{Sym^{++}(n)} \delta^2_{LE}(y, x)\nu(dx)$$

In case $\nu \in P_2(Sym^{++}(n))$, then the Karcher mean is unique (Pennec, Fillard, and Ayache 2006), noted μ and can be equivalently obtained via the following:

$$\mathbb{E}(\text{Log}_\mu(X)) = 0 \leftrightarrow \mu = \exp(E(\log(X)) \in Sym^{++}(n)$$

using the tools from Table 1 and where the second expectation, $E(\cdot)$, is now the Euclidean expectation. Depending on the type of distribution, we could treat two cases:

Discrete case We can consider a discrete distribution via the Dirac measure δ_ω and calculate the probability of X being in a set U, $P(X \in U) = \sum_{\omega \in \Omega} p(\omega)\delta_\omega(U)$, where p is the probability mass that satisfies $\sum_{\omega \in \Omega} p_\omega = 1$ with $\Omega = \{\omega_i\}_{i=1,...,n}$. Therefore,

$$E(X) = \exp\left(\sum_{\omega_i} p(\omega_i)\delta_{\omega_i} \log(X)\right) = \exp\left(\sum_{\omega_i} p(\omega_i) \log(X_i)\right) := \text{Ave}(\{X_i\}_i, \{p(\omega_i)\}_i), \quad i = 1, ..., n$$

using the notation from (Arsigny et al. 2006, Chau 2018).

Continuous case Instead of a discrete set of SPD matrices, if we consider a curve, we actually see that the expectancy, in the integral representation, along a curve $\gamma \in Sym^{++}(n)$ has the form

$$\mu = \exp \int_\gamma \log(X) d\nu(X)$$

which can actually be computed the interval of definition of the curve and the use of an appropriate Lebesgue measure.

2.3 Nonparametric wavelet estimation

From the groundbreaking works of Mallat, Meyer and Daubechies (and many others), wavelets as a mean of denoising data (ie. image, signal,...) has been extensively used for the last 30 years. As localised functions in space and frequency, which forms a big difference compared to Fourier analysis, wavelets are very suited to sparse data and can reproduce very localised features such as bumps, jumps, peaks, etc. In the present section, we give the theoretical basis for constructing wavelets transforms and implement wavelet shrinkage methods used or study in this work. The next developments allow us to build wavelets and their transforms on Riemannian manifolds equipped with the affine-invariant or the log-Euclidean metric. For the next subsections, we follow closely (Chau 2018, Klees and Haagmans 2000, Jansen 2001, G. Nason 2008).

2.3.1 One-dimensional wavelets

Because a wavelet transform acts as decomposition of a signal into different pieces, from broad to fine, of different degree of detail or resolution, it is particularly suitable for performing nonparametric estimation of a signal corrupted by noise. This multiresolution property is at the basis of the following definition:

A multiresolution analysis (MRA) in $L_2(\mathbb{R})$ is given by an increasing sequence of closed subspaces $\{V_j, j \in \mathbb{Z}\}$ in $L_2(\mathbb{R})$ satisfying the following conditions:

1. $\cap_j V_j = \{0\}$ and $\cup_j V_j$ is dense in $L_2(\mathbb{R})$
2. $\forall f \in L_2(\mathbb{R})$ and $\forall j \in \mathbb{Z}$, $f(x) \in V_j \leftrightarrow f(2x) \in V_{j+1}$
3. $\forall f \in L_2(\mathbb{R})$ and $\forall k \in \mathbb{Z}$, $f(x) \in V_0 \leftrightarrow f(x-k) \in V_0$
4. $\exists$ a function $\phi \in V_0$, called the *scaling function* of the MRA such that $\{\phi(x-k) : k \in \mathbb{Z}\}$ is a Riesz basis for V_0

If the latter basis is orthonormal, we can define $\phi_{j,k}(x) := 2^{j/2}\phi(2^j x - k)$ which forms, for $k \in \mathbb{Z}$, an orthonormal basis for V_j. An orthogonal projection P_j onto the subspace V_j is given by:

$$P_j f = \sum_k \langle f, \phi_{j,k} \rangle \phi_{j,k}$$

which corresponds to the approximation of f at the resolution scale V_j. Because $V_j \subset V_{j+1}$, we get an increased approximation of f for increasing j.

In case the basis of the point 4 of the MRA is not orthogonal, we can construct a dual MRA $\{\tilde{V}_j, j \in \mathbb{Z}\}$ in $L_2(\mathbb{R})$ and define a dual scaling function $\tilde{\phi}$, with $\tilde{\phi}_{j,k}(x) := 2^{j/2}\tilde{\phi}(2^j x - k)$, then the projection P_j can be computed as:

$$P_j f = \sum_k \langle f, \tilde{\phi}_{j,k} \rangle \phi_{j,k}$$

Looking at the projection of a function f onto the subspaces V_j and V_{j+1}, we actually see that $P_{j+1}f - P_j f$ corresponds to the difference or the detail between the two subspaces. This detail space,

noted W_j, is related by those subspaces via $V_{j+1} = V_j \oplus W_j$. Working recursively, this implies that

$$L_2(\mathbb{R}) = \bigoplus_j W_j = V_{j_0} \oplus \bigoplus_{j \geq j_0} W_j. \tag{2}$$

From the MRA defined above, there actually exists a function $\psi \in W_0$ called a *mother wavelet function*, such that $\{\psi_{jk} = 2^{j/2}\psi(2^j \cdot -k) : k \in \mathbb{Z}\}$ forms a basis of W_j. This implies that $W_j \perp \tilde{V}_j$ because $P_j V_{j+1} = V_j$ and, by duality, we can also define a dual wavelet function $\tilde{\psi}$ where we find that $\tilde{W}_j \perp V_j$. The orthogonality between (dual) detailed spaces and their non-detailed (dual) subspaces actually imply that $\langle \psi_{jk}, \tilde{\psi}_{ml} \rangle = \delta_{jm}\delta_{kl}$ (orthogonality relations) and we have:

$$(P_{j+1} - P_j)f = \sum_k \langle f, \tilde{\psi}_{jk} \rangle \psi_{j,k}$$

By (2), we can decompose any function $f \in L_2(\mathbb{R})$ as

$$f = \sum_{j,k} \langle f, \tilde{\psi}_{jk} \rangle \psi_{j,k} = \sum_k \langle f, \tilde{\phi}_{j_0 k} \rangle \phi_{j_0,k} + \sum_{j \geq j_0} \sum_k \langle f, \tilde{\psi}_{jk} \rangle \psi_{j,k} \tag{3}$$

where f is thus given by a sum of scaling and wavelet functions. Thus, we see that following the MRA previously defined, we have the scaling function ϕ and the mother wavelet ψ. Because $V_0 \subset V_1$ ($W_0 \subset V_1$), those functions can be expressed as linear combinations of the basis functions in V_1 in the following manner:

$$\phi(x) = \sum_k h_k \phi(2x - k), \quad \psi(x) = \sum_k g_k \phi(2x - k)$$

which are the two scale relations with sequences $(h_k)_k$ and $(g_k)_k$. The same type of relations exists for the dual MRA as they have the same properties. One important property shared by wavelet functions is the existence of vanishing moments. If the $(\psi_{jk})_k$ does not form an orthonormal basis, we actually have that the dual mother wavelet $\tilde{\psi}$ has N vanishing moments in the sense that:

$$\int t^k \tilde{\psi}(t) dt = 0, \quad 0 \leq k < N$$

This actually means that $\tilde{\psi}$ is orthogonal to all polynomials of degree lower than $N-1$. Looking back at the decomposition of f, we see that a locally smooth function, say p-times continuously differentiable for $p < N$, will therefore produce small wavelets coefficients $|\langle f, \tilde{\psi}_{jk} \rangle|$ that vanishes rapidly as j increases. This is important because polynomials or signals having similar properties are well approximated by a thresholded wavelet expansion if the dual basis functions have a sufficiently high number of vanishing moments.

2.3.2 Lifting scheme

In this section section, we review the construction that leads to the so-called *second generation* wavelets via the lifting scheme (Jansen and Oonincx 2005), where the wavelet transform is factorised as basic spatial *lifting* operations. This is particularly adapted for developing wavelets and wavelet transforms into more general settings, such as non-Euclidean data spaces. Again, let's take a function $f \in L_2(\mathbb{R})$, which is here observed on a equidistant grid at the locations f_j ($j \in \mathbb{Z}$). Via the lifting scheme, a discrete wavelet transform is based upon 3 operations:

1. **Split**: performs a partition of the observations into 2 disjoint sets, eg. E and O composed of observations with even and odd indices, respectively.

2. **Predict**: with the use of a prediction operator P, we make the following replacement: $O' \leftarrow O - P(E)$, where $P(E)$ takes the even indexed observations as input. If f shows some high (low) degree of regularity (ie. between ajdacent observations), then the detail $O - P(E)$ coefficients should be small (high).

3. **Update**: with the use of an update operator U, we compute: $E' \leftarrow E + U(O')$

Those operations allows to contruct an ensemble of detail (ie. differences) coefficients d_{jk} and scaling (ie. averages) coefficients s_{jk}, at the scale j. The structure of the lifting scheme allows for in-place calculations at all stages, ie. overwriting of O or E by the details or the averages of the step, does not have to be done simultanuously. Also, this approach is invertible wich allows for reconstruction of the original input. The resulting inverse wavelet transform on the coefficients d_{jk} and s_{jk} at scale j, gives us, at scale $j+1$:

$$s_{j+1,2k} \in E \leftarrow s_{jk} \in E' - U(d_{jk} \in O')$$
$$s_{j+1,2k+1} \in O \leftarrow d_{jk} \in O' + P(s_{j+1,2k} \in E)$$

As firs-step cases, we now consider interpolating subdivision and average-interpolating subdivision (Donoho 1993) as it will serve to construct the wavelet transform used in the non-Euclidean manifolds considered in the present work. In the simple case of the Haar wavelet transform, we have a constant extrapolation from the left even neighbour meaning that we have a constant interpolation. In the polynomial interpolating approach, we use the symmmetric left and right neighbouring observations to define the subdivision scheme.
In the case of the polynomial interpolation subdivision, we write $N = 2D$ ($D \geq 1$) as the order of the subdivision scheme (which is also the number of vanishing moments). Next, we consider our input data $(s_{j,k})_k \in V_j$ at scale j. Given the data $s_{j,k}$ with symmetric neighbours
$\{s_{j,k-D+1}(x_{j,k-D+1}), ..., s_{j,k}(x_{j,k}), ..., s_{j,k+D}(x_{j,k+D})\}$ where the equidistant locations are written from $x_{j,k} = k2^{-j}$. From this, the prediction of the finer scale coefficient $s_{j+1,2k1}$ from the coarser scale coefficients follows 2 steps:

1. construct the unique interpolating polynomial p of degree $N-1$, in a way that $p(x_{j,k+l}) = s_{j,k+l}$ for $-D+1 \leq l \leq D$

2. predict $s_{j+1,2k+1}$ as the value of p at $x_{j+1,2k+1}$, ie.
$P(\{s_{j,k-D+1}(x_{j,k-D+1}), ..., s_{j,k}(x_{j,k}), ..., s_{j,k+D}(x_{j,k+D})\}) = p(x_{j+1,2k+1})$

Although the interpolation subdivision is computationally quick to evaluate an interpolating polynomial at a specific location, especially with the efficient and stable Neville's algorithm , it is not well-adapted when the goal is to remove noise in a given signal. Because the scaling coefficients at scale j and $j+1$ are related by $(s_{j,k})_k \subset (s_{j+1,k})_k$, when one considers a signal corrupted by noise, the

variance of the coefficients noise will not be homogeneous from one scale to another (Rahman et al. 2005). The problem is that the classical wavelet thresholding methods rely on the homogeneous variance across scales. In the order to get a more adapted subdivision method, (Donoho 1993) proposed an average-interpolating scheme where the scaling coefficients at scale j are averages of the scaling coefficients at scale $j+1$. This means that, at coarser scales, the resulted noise gets also averaged and, therefore, the noise variance of the wavelets coefficients will be homogeneous across scales.

Average-interpolating subdivision Let's now fix $N = 2D + 1$ ($D \geq 0$), where the order of the subdivision scheme and the number of vanishing moments are now an odd number. Using the same locations $x_{j,k}$ as above, our input scaling coefficients have the following form:

$$(s_{j,k})_k = (\mathcal{I}_{j,k}(f))_k = \left(\int_{x_{j,k}}^{x_{j,k+1}} f(u)du \right)_k, \quad I_{j,k} = [x_{j,k}, x_{j,k+1}]$$

where f is a given function. The prediction of (even and odd) scaling coefficients at the scale $j+1$ as averages over $I_{j+1,2k}$ and $I_{j+1,2k+1}$ based on the polynomial with averages $\{s_{j,k-D}, ..., s_{j,k+D}\}$ over the intervals $I_{j,k-D}, ..., I_{j,k+D}$ are obtained via the following 2 steps:

1. Construct the unique average-interpolating polynomial p of degree $N-1$, $\mathcal{I}_{j,k+l}(p) = s_{j,k+l}$ for $-D \leq l \leq D$

2. Predict $s_{j+1,2k}$ and $s_{j+1,2k+1}$ as the average values of the polynomial p over $I_{j+1,2k}$ and $I_{j+1,2k+1}$, that is,
   $s_{j+1,2k} = \mathcal{I}_{j+1,2k}(p)$ and $s_{j+1,2k+1} = \mathcal{I}_{j+1,2k+1}(p)$

Computation of the predictions based on p are simplified because $P(x) = \int_0^x p(u)du$ is still a polynomial, thus, if we note $P_{j,k,N}(x) := \int_{x_{j,k-D}}^x p(u)du$, we find N interpolation conditions for $P_{j,k,N}$ that uniquely determine the average-interpolating polynomial p of degree $N-1$:

$$\begin{aligned} P_{j,k,N}(x_{j,k-D+1}) &= s_{j,k-D} \\ P_{j,k,N}(x_{j,k-D+2}) &= s_{j,k-D} + s_{j,k-D+1} \\ &\vdots \\ P_{j,k,N}(x_{j,k+D}) &= \sum_{l=-D}^{D} s_{j,k+l} \end{aligned}$$

Now, we see that we can obtain the finer scale coefficients as differences of the $P_{j,k,N}$ at specific locations:

$$\begin{aligned} s_{j+1,2k} &= P_{j,k,N}(x_{j+1,2k+1}) - P_{j,k,N}(x_{j,k}) \\ s_{j+1,2k+1} &= P_{j,k,N}(x_{j,k}) - P_{j,k,N}(x_{j+1,2k+1}) \end{aligned}$$

Again, as in the interpolating subdivision, one can use Neville's algorithm to compute the $P_{j,k,N}(x)$. Also, working on a non-dyadic grid or unequal intervals is possible, which is used in the present work

and implemented in (Chau 2018) for the subdivision scheme in the non-Euclidean context.
The average-interpolating subdivision scheme of order N defines an MRA with scaling function ϕ and mother wavelet ψ. The scaling function satisfies a number of properties, and the vanishing moments ensures that polynomials of degree at most $N-1$ can be reproduced by the average-interpolation subdivision. We also have the two-scales equations, with which we can rewrite a linear average-interpolation prediction of $s_{j+1,2k+1}$ as follow:

$$\text{Prediction}(s_{j+1,2k+1}) = \sum_{l=-D}^{D} h_{2l} s_{j,k+l}$$

where the two-scale sequences $\vec{h}_N = \{h_{-N+1}, ..., h_0, ..., h_N\}$ (associated to ϕ) are given by:

$$\vec{h}_1 = \{1, 1\}$$
$$\vec{h}_3 = \left\{-\frac{1}{8}, \frac{1}{8}, 1, 1, \frac{1}{8}, -\frac{1}{8}\right\}$$
$$\vec{h}_5 = \left\{\frac{3}{128}, -\frac{3}{128}, \frac{11}{64}, -\frac{11}{64}, 1, 1, -\frac{11}{64}, \frac{11}{64}, -\frac{3}{128}, \frac{3}{128}\right\}$$
$$\vec{h}_7 = \left\{-\frac{5}{1024}, \frac{5}{1024}, \frac{44}{1024}, -\frac{44}{1024}, -\frac{201}{1024}, \frac{201}{1024}, 1, 1, \frac{201}{1024}, -\frac{201}{1024}, -\frac{44}{1024}, \frac{44}{1024}, \frac{5}{1024}, -\frac{5}{1024}\right\}$$

for the different subdivision orders $N = \{1, 3, 5, 7\}$.
This prediction leads to the wavelet transform, with a primal and dual lifting steps. Using the same notation as before, let $(s_{j+1,k})_k = (\mathcal{I}_{j+1,k}(f))_k$. In our context the construction of the wavelet transform begins with a Haar updating step:

1. Haar updating step: $s_{j,k} \leftarrow (s_{j+1,2k} + s_{j+1,2k+1})/2$ (replacement of even averages at scale $j+1$ by averages at scale j)
2. $d_{j,k} \leftarrow s_{j+1,2k+1} - \sum_{l=-D}^{D} h_{2l} s_{j,k+l}$ (replacement of odd averages at scale $j+1$ by detail coefficients at scale j.

Again, the prediction order N determines the number of vanishing moments of the dual wavelet function. The inverse wavelet transform is then obtained by inverting the dual and primal lifting steps.

2.3.3 Linear and nonlinear wavelet thresholding methods

The wavelet transform described above is used to decompose a given one-dimensional signal f into different pieces, from coarse to fine, which is then described by wavelet coefficients. From the relation (3), we write

$$f = \sum_{j,k} \langle f, \tilde{\psi}_{jk} \rangle \psi_{j,k} = w_{j,k} \psi_{j,k}$$

where $\langle f, \tilde{\psi}_{jk} \rangle = w_{i,j}$ are the wavelet coefficients[4]. We note that if the inner product is small for a given scale j and parameter k, this suggests that there is no "similarity" between the function f and the related wavelet basis $\psi_{j,k}$. Hence, higher wavelets coefficients will be related to more variable structures (ie. jumps, bumps, peaks,...) and small (close to zero) coefficients will suggest a less variable, more regular type of function. With that in mind, we can ask ourselves: *What happens if the original signal is corrupted by noise?* Let's look at a corrupted signal f' which, in the wavelet world, can be written

$$f' = \sum_{j,k} w'_{j,k} \psi_{j,k}$$

Now, in an Euclidean additive signal-plus-noise model, we have

$$f' = f + \epsilon^w \rightarrow \sum_{j,k} w'_{j,k} \psi_{j,k} = \sum_{j,k} w_{j,k} \psi_{j,k} + \sum_{j,k} \epsilon^w_{j,k} \psi_{j,k} \rightarrow w'_{j,k} = w_{j,k} + \epsilon^w_{j,k}$$

where $\epsilon^w_{j,k}$ are wavelet coefficients of the noise (assumed to be the same among the coefficients). We see that the linearity of the wavelet transform actually gives an equivalence between the signal-plus-noise data model and its counterpart in the wavelet domain. Now, let's consider n real-valued noisy observations $\mathbf{y} = (y_i)_{i=1,\dots,n}$ from a signal-plus-noise model with $i.i.d.$ Gaussian noise $\boldsymbol{\epsilon} = (\epsilon_i)_{i=1,\dots,n} \sim N(0, \sigma^2)$:

$$\mathbf{y} = \mathbf{f} + \boldsymbol{\epsilon}$$

evaluated on an equidistant grid. As we are interested in denoising the output $\mathbf{y}$, we want to find an estimator $\hat{\mathbf{f}}$ of $\mathbf{f}$. Before discussing further methods of noise-removal using wavelets, we focus now on the efficiency of the said estimator. One way to assess the best-choice of estimators is to compute the mean square error (MSE) between the estimator and the deterministic signal $\mathbf{f}$. If we note $\mathbf{w}$ and $\hat{\mathbf{w}}$, respectively, the wavelet coefficients of $\mathbf{f}$ and $\hat{\mathbf{f}}$, we now have

$$MSE(\hat{\mathbf{f}}, \mathbf{f}) = \frac{1}{n} E||\hat{\mathbf{f}} - \mathbf{f}||_2^2 = \frac{1}{n} E||\hat{\mathbf{w}} - \mathbf{w}||_2^2 = MSE(\hat{\mathbf{w}}, \mathbf{w}),$$

where we see that assessing the efficiency of the estimator can be also be done in the wavelet domain. Note that (Jansen 2001) the noise will still be Gaussian (ie. $N(O, \sigma^2)$) in the wavelet domain, because we work in an orthogonal basis.

Small wavelet (ie. close to 0) coefficients will typically appear in case of smooth functions (either globally or with locally varying degree of smoothness), even with few number of jumps. Therefore, there will be a large amount of those coefficients and the main contribution to the coefficient will be

[4] In the following, we will take the case of an orthogonal wavelet transform for which we have $\tilde{\psi}_{jk} = \psi_{jk}$.

from the noise, as its impact covers (equally) the whole range of wavelet coefficients. Large (in absolute value) coefficients are carrying more signal information than noise and, for smooth-behaving signals, they are present in a few number compared to the small coefficients. These observations suggest[5], as a method to remove noise, to set the small coefficients of $\hat{\mathbf{w}}$ which are dominated by noise, to 0. This is called thresholding: the value of the threshold and the rule associated depend on the method used.

Linear threshdolding In the case of linear thresholding, we typically make the choice to keep all the wavelets coefficients up to some scale J_0, meaning that all the other coefficients from the finer scales $j > J_0$ are set to zero. The estimator is then written

$$\hat{\mathbf{f}} = \sum_{j>J_0} \sum_k w'_{j,k} \psi_{j,k}.$$

Because the smoothing is the same for any given location, heterogenous degrees of smoothness are not detected by this approach, which is then more adapted to globally smooth functions. For this reason and the observations made above, we consider here a nonlinear procedure.

Nonlinear threshdolding Another nonparametric estimation procedure is to fix, from the data, a level of shrinkage of the wavelet coefficients, where the noisy coefficient $w'_{j,k}$ is either rejected (ie. set to 0) if it is smaller in absolute value than a threshold $\lambda > 0$ or kept untouched[6]. This is called *hard* thresholding and the related estimator is then written as

$$\hat{\mathbf{f}} = \sum_j \sum_k w'_{j,k} \mathbb{1}_{(|w'_{j,k}|>\lambda)} \psi_{j,k}$$

We clearly see that the wavelet coefficients are either cancelled or left untouched depending on the absolute value compared to the threshold. In the present work, we will deal with hard thresholding, as a mean of denoising, but we will add another constraint based on the following observation. For a given discontinuity in a signal, on a fixed location, we will actually have wavelet coefficients that are impacted across several scales at the said location. The idea is to impose a specific *tree-structure* where the nonzero wavelet coefficients $w'_{j+1,2k}$ and $w'_{j+1,2k+1}$ (from the hard thresholding) implies that $w'_{j,k}$ also has to be nonzero (Freyermuth 2011). The noise related wavelet coefficients, as they will be set to 0, won't be a part of the tree. In this work, we will perform tree-structured wavelet thresholding in the context of SPD matrix-valued curves with the log-Euclidean metric. A more detailed explanation of the overall above procedure is given in (Chau 2018).

[5]Wavelets coefficients of relatively smooth functions actually decay fast across scales (**Walnut2002**), thus at the finest scales there is mostly noise than signal.

[6]It can also be adjusted in value by the threshold in the case of *soft* thresholding, which we don't cover here.

3 Data models in the log-Euclidean framework

Many of the data models for symmetric and positive-definite matrices, which combine a suggested deterministic signal and specific noise that corrupts the signal, are mainly showing a linear or additive (Lin, Muller, and Park 2020) function between signal and noise. The additive link between signal and noise is of course meant as an extrapolation of the Euclidean additive model, the so-called *signal-plus-noise* model, within the intrinsic manifold at stake. In this section, we will look two types of data generating processes and the possible link between them.

3.1 Simple additive signal-plus-noise model

The principal data model used in the present work is a typical signal-plus-noise model in the log-Euclidean manifold. We suppose that we have a function $\gamma : [0,1] \to Sym^{++}(d)$ which represents a (know or unknown) deterministic signal, such that $\gamma(k/n)$ are represented on a grid, and random variables $\xi_k \in Sym(d)$ which are i.i.d. mean-zero variables with distribution $\tilde{\nu} \in P_2(Sym(d))$. The intrinsic signal-plus-noise model in the log-Euclidean manifold writes as:

$$X_k = \mathrm{Exp}_{\gamma_k} \Gamma_{Id}^{\gamma_k}(\xi_k) = \exp(\log \gamma_k + \xi_k), \quad k = 0, ..., n \tag{4}$$

This is the main data model used in the following chapters. As from the relation (1), we actually see that taking the logarithm on both sides of it, we indeed get a Euclidean relation on the right side. We will suppose that the noise random matrices ξ_k have independant entries with homogeneous noise, meaning that the variance of ξ_k is:

$$\mathrm{Var}(\xi_k) = \sigma^2 \mathbf{1}_{d \times d}$$

which basically assumes that the noise will affect all the components of the matrix-valued signal in the same way. This is the same type of hypothesis made in the wavelet context of the last section.
The above model doesn't show what is the underlying noise distribution. A possible choice are the Riemannian Gaussian distributions (Said et al. 2017) but, in our case, the most natural distribution $\tilde{\nu}$ to use here are actually Wishart's distributions (as in (Chau 2018) for HPD matrices) which is a family of distributions defined over symmetric, nonnegative-definite random matrices. Although it was necessary in the case of (Chau 2018) because a multitapering approach was used to make sure that the observations are indeed HPD, it leads to biased estimator and, therefore, implies that a bias-correction needs to be implemented. In the SPD case, this is not necessary and we justify the use of the latter model, without considering Whishart distributions, with the following. Recall that the link between the set of symmetric $d \times d$ matrices and the set of symmetric and positive-definite $d \times d$ matrices:

$$Sym^{++}(d) \cong \exp(Sym(d)), \quad Sym(d) \cong \log(Sym^{++}(d))$$

Based on a Central Limit Theorem in the log-Euclidean manifold (Schwartzman 2014), we actually have

$$\log X - \log M \sim N(0, \Sigma)$$

with $X, M, \Sigma \in Sym^{++}(d)$. Thus, we can generate, with the variance of ξ_k, random symmetric matrices using our model (4) because

$$\log X_k - \log \gamma_k = \xi_k \sim N(0, \Sigma).$$

Then, under exponentiation, we get back random SPD matrix-valued variables.
In the next section, we will look at the spectral decomposition of SPD matrices and, based on a geometrical argument, propose another way to use this representation of data that includes both a deterministic (ie. signal) and a random (ie. noise) parts.

3.2 Spectral decomposition of SPD matrices

As a simple and perhaps most representative case, we look at a real 2×2 SPD matrix A, hence having the following properties: $A^t = A$ and $x^t A x > 0$, $\forall x \in \mathbb{R}_0^{2\times 2}$. The matrix A being *symmetric* implies that the eigenvalues of A are all real and that each eigenvector, corresponding to a distinct[7] eigenvalue, are orthogonal to one another. This leads, following the classic spectral theorem, to the next decomposition:

$$A = Q\Lambda Q^{-1} = Q\Lambda Q^t$$

where Q is an orthogonal matrix (of the eigenvectors), ie. $Q^t = Q^{-1}$, and Λ is a diagonal matrix (of the real eigenvalues). Now, because we are dealing with covariance-type of matrices, we have to take in account the *positive definite* (PD) property of A. This property implies that the 2 eigenvalues of A are strictly positive. On a side note, we can also apply the *Cholesky decomposition* when one deals with SPD matrices, that is, A can be written as $A = L^t L$ where L is a triangular inferior matrix.
The positive-definiteness actually allows, together with the spectral decomposition, to give a geometric representation (or figure) of a given PD matrix. Let's consider the a generic 2×2 matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad a, b, c, d \in \mathbb{R}$$

where $A^t = A$ implies that $b = c$ and the PD-condition implies:

$$ax_1^2 + 2bx_1x_2 + cx_2^2 > 0, \quad (x_1, x_2) \in \mathbb{R}_0^2$$

With the use of the spectral decomposition, we can rewrite this expression as

$$\begin{aligned}
(x_1, x_2)^t (Q\Lambda Q^t)(x_1, x_2) &= ((x_1, x_2)^t Q)\Lambda(Q^t(x_1, x_2)) \\
&= (Q^t(x_1, x_2))^t \Lambda (Q^t(x_1, x_2)) \\
&= (y_1, y_2)^t \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} (y_1, y_2), \quad (y_1, y_2) = Q^t(x_1, x_2) \\
&= \lambda_1 y_1^2 + \lambda_2 y_2^2 > 0
\end{aligned}$$

[7] Of course, a given eigenvalue might have a multiplicity bigger than one.

where $\lambda_1, \lambda_2 > 0$ and the overall expression has to be positive because of the PD constraint. We have an ellipse's equation that is expressed in the orthogonal basis of the eigenvectors:

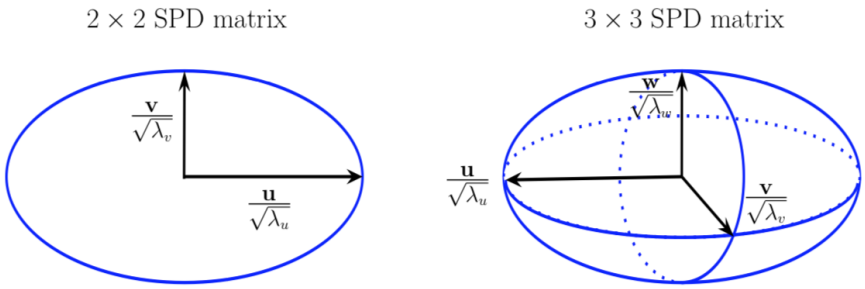


Figure 4: Ellispoid's representation of a 2×2 and 3×3 SPD matrices, with $\mathbf{u}, \mathbf{v}, \mathbf{w}$ being the eigenvectors and the related λ's are the eigenvalues.

The principal axes of the ellipse are actually the directions of the eigenvectors, while the length of these axes are related to the eigenvalues. We get an ellipsoid for $n = 3$ and, in the case $n = d$, we find the equation of an ellipsoid with an intrinsic geometry of dimension $d - 1$, forming an hypersurface of $\mathbb{R}^d$.

Now that we identified the geometric representation of a given matrix $A \in Sym^{++}(d)$, we look at how we can translates this observation into a data model. Any deformation of a SPD matrix, meaning any noisy corruption of the original SPD data, should then be seen as a deformation of the ellipsoid. The following changes, in the spectral decomposition domain of the SPD matrix, can be applied to deform a SPD matrix into another SPD matrix:

- scaling of the eigenvalues, ie. stretching (or shrinking) the ellipsoid along principal axes;
- rotation of the eigenvectors, ie. rotation of the ellipsoid's principal axes.

We start from a $d \times d$ matrix $A = Q\Lambda Q^t$, where the columns of $Q \in SO(d)$ (which is the set of $d \times d$ rotation matrices) are orthogonal (eigen)vectors of A, and $\Lambda \in Diag^{++}(d)$ is a diagonal matrix of (strictly) positive eigenvalues. These observations could be useful to describe the way noise can impact SPD matrices, especially in DTI (Pasternak, Sochen, and Basser 2010), because the diffusion tensor in a 3×3 SPD matrix. From (Jung and Schwartzman 2014), we can actually write a curve of SPD matrices, starting from A, containing continuous transformation of this initial matrix[8]:

$$f(t) = \exp(At)Q\Lambda \exp(Lt)Q^t \exp(-At) \in Sym^{++}(d), \forall t \in \mathbb{R} \tag{5}$$

showing a combination of stretching/shrinking and rotations of the ellipsoid. Here, the scaling transformation $\Lambda \to \Lambda \exp(Lt)$ ensures that the transformation stays in $Diag^{++}(d)$. The rotation of the eigenvectors, at a constant angular rate, is described by $\exp(At)Q$, where A is an antisymmetric matrix with $\exp(At)Q(\exp(At)Q)^t = \exp(At)QQ^t \exp(-At) = Id$. We can adapt this to develop a data

[8]With the following: $A^t = -A$ and $L = \text{diag}(l_1, ..., l_d) \in Diag(d)$.

model in the log-Euclidean manifold. Eventhough the model should make sure that noise corrupts the overall matrix (ie. impacting eigenvalues and eigenvectors), we will make an important assumption to ease the following developments as they serve mainly as an introduction to a possible area of research. Namely, we will suppose that eigenvectors are left untouched by noise which obviously is a too strong assumption as it contradicts what is observed, see for eg. DTI analysis.
Using the same notation as before, let's write the deterministic signal using the spectral decomposition:

$$\gamma_k = Q_k \Lambda_k^\gamma Q_k^T, \quad k = 0, ..., n$$

An observed sequence of SPD matrices is then written as

$$X_k = Q_k \Lambda_k^X Q_k^T, \quad k = 0, ..., n$$

where the orthogonal rotation matrices Q_k are fixed (for each k). The noise is thus corrupting the eigenvalues, thus generating stretching of shrinking along the principal axis of the original ellipsoids. As we have seen before, an available choice would be:

$$\begin{aligned}
\Lambda_k^X = \exp(L)\Lambda_k^\gamma \leftrightarrow \begin{pmatrix} \lambda_1^X & 0 & \\ 0 & \lambda_2^X & 0 \\ 0 & 0 & \lambda_3^X \end{pmatrix} &= \begin{pmatrix} e^{l_1}\lambda_1^\gamma & 0 & \\ 0 & e^{l_2}\lambda_2^\gamma & 0 \\ 0 & 0 & e^{l_3}\lambda_3^\gamma \end{pmatrix} \\
&= \begin{pmatrix} \lambda_1^\gamma(1 + l_1 + l_1^2/2 + ...) & 0 & \\ 0 & \lambda_2^\gamma(1 + l_2 + l_2^2/2 + ...) & 0 \\ 0 & 0 & \lambda_3^\gamma(1 + l_3 + l_3^2/2 + ...) \end{pmatrix} \\
&= \Lambda_k^\gamma + \begin{pmatrix} \lambda_1^\gamma(l_1 + l_1^2/2 + ...) & 0 & \\ 0 & \lambda_2^\gamma(l_2 + l_2^2/2 + ...) & 0 \\ 0 & 0 & \lambda_3^\gamma(l_3 + l_3^2/2 + ...) \end{pmatrix} \in Diag^{++}(3)
\end{aligned}$$

where we took a 3×3 matrix for convenience and we used the Taylor development of the exponential function. A more tractable model can be used by suggesting that the noise impact each diagonal component in the same way:

$$\Lambda_k^X = \begin{pmatrix} \lambda_1^X & 0 & 0 \\ 0 & \lambda_2^X & 0 \\ 0 & 0 & \lambda_3^X \end{pmatrix} = \begin{pmatrix} \lambda_1^\gamma \times e^l & 0 & 0 \\ 0 & \lambda_2^\gamma \times e^l & 0 \\ 0 & 0 & \lambda_3^\gamma \times e^l \end{pmatrix} = e^l \Lambda_k^\gamma$$

which ensures that the eigenvalues are all positive: $(\lambda_i^X)_{i=1,2,3}) > 0$, as needed. This leads to the following data model:

$$X_k = Q_k \Lambda_k^X Q_k^T = Q_k (e^l \Lambda_k^\gamma)_k Q_k^T = e^l \gamma_k = \exp(\log \gamma_k + l \times Id_k)$$

Thus, we find again an additive signal-plus-noise model in the log-Euclidean manifold with the noise parameter being $l \in \mathbb{R}$. This way we clearly see that the impact of noise will only stretch or shrink the original data eigenvalues, while preserving the PD property. The distribution choice of the univariate

random variable l is still to define to account for smaller/larger values of the original eigenvalues. As this section merely serves as an introduction on the subject and because we actually get a special case of the data model from section 3.1, we won't show the results of the small simulation setup in chapter 6. We stress that the noise choice above clearly is a big restriction and does not reflect what is observed in many areas of research where the data comes from SPD matrix-valued observations. Still, as shown in the relation (5), we could generalize this approach by including an impact on the eigenvectors (Jung and Schwartzman 2014).

4 The intrinsic average-interpolation scheme in the log-Euclidean manifold

After a description of the average-interpolation subdivision scheme, within the log-Euclidean framework, we have a look at the resulted wavelet transforms. Then, we discuss the behaviour of the log-Euclidean and affine-invariant estimators under transformation from specific groups of symmetry.

4.1 The average-interpolation refinement scheme adapted to the log-Euclidean metric

In this first part, we review the refinement procedure, established in (Chau 2018) in the context of the affine-invariant manifold, adapted to the log-Euclidean manifold. Basically, we will exploit the simplification of the (differential) geometrical tools with the log-Euclidean metrics, but also the use of the lifting scheme, adapted here to a non-Euclidean manifold. With this intrinsic average-interpolation (AI), we deal with wavelets within a refinement scheme that gives a multiresolution representation. The predicting step (ie. the prediction of midpoints) is done via the intrinsic AI, going from broad to fine scales. The found (fast) wavelet transforms, forward and backward transforms, are allowing us to obtain estimated locally averages at different scales from observed or simulated data. Like the scalar form presented in section 2.3.2, the idea is to obtain a subdivision scheme allowing for a clear and useful (inverse)-wavelet transform in the context of SPD-matrix valued data. We note that the following is an adaptation of (Donoho 1993)'s approach on the real line.

Recall that we work with curves of $d \times d$ SPD matrices, living in $Sym^{++}(d)$. The following approach is *intrinsic* because it works directly in the manifold and uses a geodesically-adapted version of the average-interpolation seen in section 2.3.2. A few constraints need to be take in account. The following refinement scheme not only has to allow for noise removal in the coarser scales but also has to make possible the reconstruction of (intrinsic) polynomial of a given degree. The first constraint is met via a weighted average in the coarser scales (from the finest scale) and the second one is a typical property met in classical wavelets theory (Hinkle, Fletcher, and Joshi 2014). The tools used can be found in the section 2.2.

For our data, we consider a curve of SPD matrices $\gamma : \mathcal{I} = [0,1] \to Sym^{++}(d)$, which is square integrable because $\int_0^1 ||\gamma(u)||_F^2 du < \infty$. We look at $n = 2^J$ data points, defined on a finite-dimensional equally spaced grid, at the resolution scale J. This means that we sample 2^J observations $\gamma(k/2^J)_{k=0,\dots 2^J-1}$, corrupted by noise, from the γ curve. A practical choice in the context of wavelets transform is to use a dyadic number of points and define the input to be the midpoints (ie. scaling) coefficients $(M_{J,k})_k$ which correspond to locally intrinsic averages over the equally sized and non-overlapping intervals $I_{J,k}$. These intervals cover the whole definition domain of γ as $\cup_k I_{J,k} = [0,1]$. Within the Riemannian manifold equipped with the log-Euclidean metric, we have

$$M_{J,k} = \text{Ave}_{I_{J,k}}(\gamma) = \exp \int_\gamma \log(x) d\nu(x)$$

where we can make a change in variables using *Lebesgue integral theorem* which allows to write:

$$\int_{\gamma} \log(x) d\nu(x) = \int_{I_{J,k}} (\log \circ \gamma) |J_{\gamma}| d\nu = \int_{I_{J,k}} \frac{1}{\nu_{I_{J,k}}} \log \gamma(t) dt$$

with the Jacobian (evaluated at ν) and the relation between real Lebesgue measures (Lee 2013). Here, the value of $\nu_{I_{J,k}}$ actually corresponds to the length of the interval $I_{J,k}$, being the same for all $(I_{J,k})_k$. For the first predicting step, we build a redundant midpoint pyramid (Rahman et al. 2005) from the finest coefficients $M_{J,k}$. At the next coarser scale ($j = J - 1$), we build the following coefficients by considering the halfway point of the geodesic between the 2 points in $Sym^{++}(d)$:

$$\begin{aligned} M_{j,k} &= \text{Ave}(\{M_{j+1,2k}, M_{j+1,2k+1}\}) \\ &= \exp\left(\frac{1}{2}\log M_{j+1,2k} + \frac{1}{2}\log M_{j+1,2k+1}\right) \quad k = 0, ..., 2^J - 1 \end{aligned} \tag{6}$$

This coarsening is to be done until the level $j = 0$ where, at each level, we have a set of 2^j points.
To correctly reproduce intrinsic polynomials from a discrete set of coarse-scale midpoints $M_{j,k}$, we apply the intrinsic AI scheme to get, as an output, the finer-scale predicted midpoints $(\tilde{M}_{j+1,k})_k$. Doing so, we get an intrinsic polynomial $\tilde{\gamma}$ with $j + 1$ predicted midpoints with the scale j original midpoints. As we said, the reconstruction of intrinsic polynomial can be done using Neville's algorithm (Chau 2018) where, in our case, we need to replace the linear interpolation by geodesic interpolation[9].
To predict the midpoints, using the average-interpolation subdivision (see section 2.3.2), we compute the intrinsic cumulative mean of $\tilde{\gamma}$:

$$M_{y_0}(y) = \text{Ave}_{[y_0,y]}(\tilde{\gamma})$$

The cumulative intrinsic mean of an intrinsic polynomial of order r is again an intrinsic polynomial of order $\leq r$. Again, this is analogous to the Euclidean setting, where an integrated polynomial remains a polynomial.

Prediction away from the boundary Using the same notation as in section 2.3.2, let the order of the subdivision scheme $N = 2L+1 \geq 0$ corresponding to the collected neighbours, we want to predict the midpoints $\{M_{j,2k}, M_{j,2k+1}\}$ starting from the $(j-1)$-scale midpoints $\{M_{j-1,k-L}, ..., M_{j-1,k}, ...M_{j-1,k+L}\}$:

1. fit (via Neville's algorithm) an intrinsic polynomial $\hat{M}_{(k-L)2^{-j-1}}(y)$ of order $N-1$ passing through $\bar{M}_{j-1,0}, ..., \bar{M}_{j-1,N-1}$ (N known points) with the cumulative average
   $\bar{M}_{j-1,l} = \text{Ave}(M_{j-1,k-L}, ..., M_{j-1,k-L+l})$
   Remark: $M_{(k-L)2^{-j-1}}((2k+1)2^{-j})$ actually lies on the geodesic between $\bar{M}_{j-1,L}$ and the midpoint $M_{j,2k+1}$

2. Replacing $M_{(k-L)2^{-j-1}}((2k+1)2^{-j})$ by its estimate $\hat{M}_{(k-L)2^{-j-1}}((2k+1)2^{-j})$, we can compute the following predicted midpoint:

[9] From $P_0, ..., P_n \in \mathcal{M} = Sym^{++}(d)$ with $x_0 < ... < x_n \in \mathbb{R}$, we let $p_{i,i}(x) := P_i$ for $x \in \mathbb{R}$, $i = 0, ..., n$. We then define the geodesic interpolation as: $p_{i,j} = \exp\left(\frac{x_j - x}{x_j - x_i} \log p_{i,j-1}(x) - \frac{x - x_i}{x_j - x_i} \log p_{i+1,j}(x)\right)$ with the log-Euclidean metric.

$$\tilde{M}_{j,2k+1} = \eta(\bar{M}_{j-1,L}, \hat{M}_{(k-L)2^{-j-1}}((2k+1)2^{-j}), -2L)$$
$$= \exp\left((1+2L)\log \bar{M}_{j-1,L} - 2L \log \hat{M}_{(k-L)2^{-j-1}}((2k+1)2^{-j}\right)$$

3. For the other predicted midpoint, we know that

$$M_{j-1,k} = \text{Ave}(\{\tilde{M}_{j,2k}, \tilde{M}_{j,2k+1}\})$$
$$= \exp\left(\frac{1}{2}\log \tilde{M}_{j,2k} + \frac{1}{2}\log \tilde{M}_{j,2k+1}\right) \quad k = 0, ..., 2^J - 1 \tag{7}$$

In the scalar version of the AI subdivision, the predicted j-scale scaling coefficients are obtained via polynomial average-interpolation of the $(j-1)$-scale scaling coefficients. Those are equivalent to weighted linear combinations of the input scaling coefficients, with weights depending on the average-interpolation order N. For the intrinsic version of Neville's algorithm, we need to change the Euclidean (ie. linear here) interpolation to a non-Euclidean (ie. geodesic) interpolation. Doing so, we find that the predicted midpoints $\{M_{j,2k}, M_{j,2k+1}\}$ are still weighted averages of the inputs $\{M_{j1,kL}, ..., M_{j1,k}, ..., M_{j1,k+L}\}$, with the same weights used in the Euclidean case. Of course, the weighted averages are now intrinsic weighted averages in the Riemannian manifold. Those weights are written

$$C_{N,2l+N} \quad \text{with } -L \leq l \leq L \quad \text{and } \sum_l C_{N,2l+N} = 1$$

and are equivalent to the filter weights in the scalar average-interpolation subdivision scheme from section 2.3.2. We will focus on the case with the refinement order $N = 5$. Let's now turn to the wavelet transform associated to the intrinsic AI subdivision scheme.

4.2 Forward/backward intrinsic AI wavelet transform

Here, we give the forward/backward wavelet transforms, starting from the input data $(M_{J,k})_k$ where J is the finest scale and $k \in [0, 2^j - 1]$. The wavelets transforms are based on the midpoints pyramid $(M_{j,k})_{j,k}$, $0 \leq j \leq J$, $0 \leq k \leq 2^j - 1)$) described above. The two steps forward and backward allow to go from j-to-$(j-1)$ scale and $(j-1)$-to-j scale, respectively.

Forward wavelet transform From the intrinsic average-interpolation subdivision scheme described above, this leads to an intrinsic AI wavelet transform passing from the j-scale midpoints $(M_{j,k})_k$ (*input*) to the $(j-1)$-scale midpoints $(M_{j,k})_k$ plus j-scale $(D_{j,k})_k$ wavelet coefficients (*output*), via the following:

1. **Predict:** Compute the $(j-1)$-scale midpoints $(M_{j-1,k})_k$ through the relation (6), select a refinement order $N \geq 1$ and compute the $(\tilde{M}_{j,k})_k$ based on the 3 substeps of the prediction step above.

2. **Difference:** Based on the true and predicted midpoints at scale j, define the wavelet coefficients as an intrinsic difference
$D_{j,k} := 2^{-j/2}\text{Log}_{\tilde{M}_{j,2k+1}}(M_{j,2k+1}) \in T_{\tilde{M}_{j,2k+1}}(Sym^{++}(d))$ but we will mainly consider *whitened* wavelets coefficients as a mean to compare those coefficients across all scales and locations, by performing a parallel transport to the identity of the wavelet coefficients $D_{j,k}$:

$$\mathcal{D}_{j,k} = \Gamma^{Id}_{\tilde{M}_{j,2k+1}}(D_{j,k}) = 2^{-j/2}(\log M_{j,2k+1} - \log \tilde{M}_{j,2k+1}) \in T_{Id}(Sym^{++}(d)) \subset Sym(d)$$

where the (log-Euclidean) norms of $D_{j,k}$ and $\mathcal{D}_{j,k}$ are the same.

The whitened wavelet coefficients will play an important role in the following chapters, serving as a basis to find the rule to perform wavelet thresholding.

Backward wavelet transform This is the inverse of the forward case, exchanging the roles of input and output (which are the j-scale midpoints $(M_{j,k})_k$) with the following steps:

1. **Predict:** With an order $N \geq 1$, generate the predicted midpoints $(\tilde{M}_{j,2k+1})_k$ and compute the (j)-scale midpoints $(M_{j-1,k})_{k=0,\dots,2^j-1}$ at the odd locations $2k+1$ via:
$M_{j,2k+1} = \text{Exp}_{\tilde{M}_{j,2k+1}}(2^{j/2}D_{j,k})$.

2. **Complete:** The j-scale midpoints at even locations $2k$ are obtained from $M_{j-1,k}$ and $M_{j,2k+1}$ through the relation (7): $M_{j,2k} = \exp\left(2\log(M_{j-1,k}) - \log(M_{j,2k+1})\right)$

If we repeat the reconstruction procedure up to the scale J, we retrieve the original input sequence of local averages $(M_{J,k})_k$ for $k = 0, ..., 2^J - 1$. Note that the wavelet coefficients are stored inside $D_{j,k}$ for each scale-location (j, k). This means that, in our nonparametric estimation approach, we will look for an estimate $\hat{D}_{j,k}$ of $D_{j,k}$. In the following section, we will look at the symmetry properties of the estimator and exhibit the difference between the log-Euclidean and the affine-invariant case.

4.3 Estimator's equivariance for the log-Euclidean and the affine-invariant metrics

An important feature of our estimator is that it has to be symmetric and positive definite. The procedure described above provides a way to, from an observed of SPD matrix-valued curve, obtain an estimation of the original curve γ that lives on the same intrinsic geometry as the original data lives in. Another aspect of the resulted estimator is its symmetry-related properties. Before looking into this, let's recall that the affine-invariant Riemannian manifold and the log-Euclidian Riemannian manifold have a different metric. Therefore, as in section 2.2, the induced distances are different and

have the following invariances:

$$\begin{aligned}
\delta_R(a*p_1, a*p_2) &= \delta_R(p_1,p_2), \quad \forall a \in \mathrm{GL}(d,\mathbb{C}), \forall p_1,p_2 \in \mathcal{M}_R \quad \text{(a)} \\
\delta_{LE}(a*p_1, a*p_2) &= ||\log(a^t p_1 a) - \log(a^t p_2 a)||_F, \quad \forall a \in \mathrm{GL}(d,\mathbb{R}|aa^t = Id), \forall p_1,p_2 \in \mathcal{M}_{LE} \quad \text{(b)} \\
&= ||a^t \log(p_1)a - a^t \log(p_2)a||_F = ||a^t(\log(p_1) - \log(p_2))a||_F \\
&= \delta_{LE}(p_1,p_2)
\end{aligned}$$

where the isometry is $x \to a*x = a^*xa$ and $\mathrm{GL}(d,\mathbb{K})$ is the general linear group of $d \times d$ complex (if $\mathbb{K} = \mathbb{C}$) or real (if $\mathbb{K} = \mathbb{R}$) invertible matrices. In the log-Euclidean case, there is a restriction for orthogonal, or unitary matrices if we were within the HPD framework, matrices belonging to the $O(d)$ symmetry group. We see that the group of isometries is more general in with affine-invariant metric, being important in the context of spectral estimation. Those invariances, or isometries, will actually influence the way we interpret our estimator in case of, for example, we apply a rotation O onto our data. This can typically happen in diffusion tensor imaging, when rotation of the B-matrix is performed to make corrections due to the subject motion (Leemans and Jones 2009).
If we note D^X a noisy matrix-valued wavelet coefficient and $\hat{D}^X$ its estimated (via thresholding) version, then $O^t D^X O$ should be estimated by $O^t \hat{D}^X O$ as the component-specific thresholds are just rotated in the same manner as the coefficients components. If $(X_l)_l$ is a sequence of noisy SPD matrices and $(\hat{\gamma}_l)_l$ its wavelet-denoised estimate then $(O^t X_l O)_l$ has the following denoised counterpart: $(O^t \hat{\gamma}_l O)_l = (O * \hat{\gamma}_l)_l$ for all $O \in \mathrm{GL}(d,\mathbb{R}|OO^t = Id)$. This will obviously be the case if with have only one smoothing parameter, coming from a scalar thresholding rule.
The choice of the thresholding rule will however have a direct impact on those properties. In (Chau 2018), a trace-thresholding rule is applied based on the fact that the trace of the whitened wavelet coefficients develops into a scalar signal-plus-noise model within the context spectral estimation in the affine-invariant manifold. This results into an estimator of the following form:

$$\hat{D}^X_{j,k} = g(\mathrm{Tr}(\mathcal{D}^X_{j,k}))D^X_{j,k}$$

where $g(\cdot)$ is a real smooth function. Under the congruence $A * X$ for $A \in \mathrm{GL}(d,\mathbb{C})$, we find that:

$$\hat{D}^{X,A}_{j,k} = A^* \hat{D}^X_{j,k} A = A * \hat{D}^X_{j,k}$$

where $\hat{D}^{X,A}_{j,k}$ is the estimated wavelet coefficients for the "rotated" data. The equivariance under a congruence transformation from any invertible matrix is **only** true here because the estimator actually depends on the trace of the whitened coefficients. For the thresholding rule we will consider in the next chapters, where we take the sum of the matrix elements, we have to check whether the estimator

shows the equivariance property under transformation by any invertible and orthogonal real matrix:

$$\begin{aligned}\sum_{a,b}[\mathcal{D}^{X,A}_{j,k}]_{a,b} &= \sum_{a,b}[2^{-j/2}(\log M^{X,A}_{j,2k+1} - \log \tilde{M}^{X,A}_{j,2k+1})]_{a,b} \\ &= \sum_{a,b}[2^{-j/2}(\log A^t M^{X}_{j,2k+1} A - \log A^t \tilde{M}^{X}_{j,2k+1} A)]_{a,b} \\ &= \sum_{a,b}[2^{-j/2}(A^t \log M^{X}_{j,2k+1} A - A^t \log \tilde{M}^{X}_{j,2k+1} A)]_{a,b} \\ &= \sum_{a,b}[A^t \mathcal{D}^{X}_{j,k} A]_{a,b} \neq \sum_{a,b}[\mathcal{D}^{X}_{j,k}]_{a,b}.\end{aligned}$$

To clearly see the last inequality, let's take the following SPD matrix $S = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ and the following orthogonal matrix $A = \left(\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right)$. We find:

$$\sum_{a,b}[S]_{a,b} = 5 \neq \sum_{a,b}\left[\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\right]_{a,b} = \sum_{a,b}\left[\begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}\right]_{a,b} = 1$$

Thus, in the log-Euclidean case, with the thresholding rule applied in the following section, we cannot keep the equivariance of the estimator under an orthogonal congruence transformation. Nevertheless, apart from the fact that we deal with SPD matrices (ie. not Hermitian), we will notice that estimators, for the different thresholding rules under the log-Euclidean and the affine-invariant metrics, actually share on a general ground the same level of equivariance properties. Indeed, we see that the estimator doesn't heritate directly from the invariance property (under general linear congruence transformations) in the case of the affine-invariant metric. In (Chau 2018), it is true because of the choice of the thresholding based on the trace operator. For the log-Euclidean metric, we have the unitary congruence invariance but we see that the equivariance of the estimator is also not directly implied and will depend on the choice of thresholding rule. Overall, the general level of equivariance shared by the estimators in our framework and in the case treated by (Chau 2018) is the equivariance under unitary congruence transformations. As seen in section 2.2, we have for both metric an invariance under matrix inversion and this is particularly desirable in analysis of DTI data, where both large and small diffusions are unlikely (Quang and Vittorio 2018).

5 Wavelet thresholding within the log-Euclidean manifold

In the last section, we have seen how to construct a valid wavelet transform that can be used on non-Euclidean data, ie. SPD matrices for our case. As in the section 2.3.3, we want to implement a non-linear method to perform nonparametric estimation for corrupted by noise SPD matrices using thresholding in the wavelet coefficients domain. In (Chau 2018), they derived the wavelet coefficient decay and linear wavelet thresholding convergence rates in the context of the intrinsic average-interpolation wavelet transforms for intrinsically smooth curves of HPD matrices. They found that it coincides with the usual scalar wavelet coefficient decay and linear thresholding convergence rates on the real line (Antoniadis 1997). This will not be a point of discussion here because the same type of convergence is directly implied given the data at stake. Next, we look at how our simple signal-plus-noise log-Euclidean model translates into the wavelet domain.

5.1 Wavelet thresholding in the log-Euclidean data model

Given an observed sequence, ie. a noisy version of a real (or deterministic) signal, of $d \times d$ SPD matrices, we want to apply the following procedure in order to denoise our observed signal:

1. Apply the forward intrinsic AI wavelet transform to the observed signal
2. Threshold (or shrink) the coefficients in the intrinsic wavelet domain, depending on a specific rule
3. Apply the backward intrinsic AI wavelet transform to the modified coefficients.

As explained before, a given irregularity in the data, such as a bump or a cusp, might have an impact on all the components of a matrix of wavelet coefficients and, therefore, we should threshold all those components, at a given scale-locations, at the same time. Here, we perform a keep-or-kill approach, ie. meaning that we set to 0 the coefficients that are thresholded and, as in 2.3.3, we consider a dyadic tree-structured thresholding of wavelet coefficients.
Because of the simpler tools and the flexibility we have regarding symmetry constraints, calculations in the log-Euclidean manifold provide more possibilities regarding the thresholding rule. Indeed, let's go back to our main selected data model:

$$X_k = \exp(\log \gamma_k + \xi_k), \quad k = 0, ..., 2^J - 1 \tag{8}$$

where the number of observations is $n = 2^J$, γ_k is our deterministic signal and ξ_k are the i.i.d. mean-zero noise random variables (in $P_2(Sym^{++}(d))$. Given that our data is $X_k = M_{J,k}$, we find that the

logarithm of the midpoint relation (6) is expressed as

$$\begin{aligned}
\log M_{j,k} &= \text{Ave}(\{M_{j+1,2k}, M_{j+1,2k+1}\}) = \log\exp\left(\frac{1}{2}\log M_{j+1,2k} + \frac{1}{2}\log M_{j+1,2k+1}\right) \\
&= \frac{1}{2}\left(\log M_{j+1,2k} + \log M_{j+1,2k+1}\right) \\
&= \frac{1}{2}\frac{1}{2}\left(\log M_{j+2,2k+1} + \log M_{j+2,2k+2} + \log M_{j+2,2k+3} + \log M_{j+2,2k+4}\right) \\
&= ... = \frac{1}{2^{J-j}}\sum_{l=0}^{2^{J-j}-1}\log(M_{J,2k^{J-j-1}+l+1}) \quad (\text{with } X_k = M_{J,k}) \\
&= \frac{1}{2^{J-j}}\sum_{l=0}^{2^{J-j}-1}\left(\log(\gamma_{2k^{J-j-1}+l+1}) + \log(e^{\xi_{2k^{J-j-1}+l+1}})\right) \\
&= \log M^{\gamma}_{j,k} + \log M^{\xi}_{j,k} \qquad (9)
\end{aligned}$$

where the last step is again from the midpoint relation (6). We see that the logarithm of the midpoint $M_{j,k}$ can actually be written as a sum of logarithm of the (deterministic) signal midpoint and the (random) noise midpoint, at the same scale. We also notice that the noisy midpoint actually corresponds to the exponential of the random variable ξ, which follows a normal (multivariate) distribution (see section 3.1). Again, the $\log(\cdot)$ function flattens the curved geometry of the SPD matrices and gives us a simple Euclidean addition in the relation (9). Keep in mind that we want to exhibit the relation between the observed withened wavelets coefficients $\mathcal{D}^{X}_{j,k}$, the deterministic whitened wavelets coefficient $\mathcal{D}^{\gamma}_{j,k}$ and the random-noise whitened wavelets coefficient $\mathcal{D}^{\xi}_{j,k}$. Those coefficients are obtained by

calculating the predicted midpoints $\tilde{M}_{j,2k+1}$ as follow:

$$\begin{aligned}
\log \tilde{M}^X_{j,2k+1} &= \log \mathrm{Ave}\left(\{M^X_{j-1,k+l}\}_{l\in[-L,L]}; \{C_{N,2l+N}\}_{l\in[-L,L]}\right) \\
&= \log \mathrm{Exp}_{\tilde{M}^X_{j,2k+1}}\left(\sum_{l=-L}^{L} C_{N,2l+N}\mathrm{Log}_{\tilde{M}^X_{j,2k+1}}(M^X_{j-1,k+l})\right) \\
&= \log\exp\left\{\log \tilde{M}^X_{j,2k+1} + d_{\tilde{M}^X_{j,2k+1}}\log \times \sum_{l=-L}^{L} C_{N,2l+N}\mathrm{Log}_{\tilde{M}^X_{j,2k+1}}(M^X_{j-1,k+l})\right\} \\
&= \log \tilde{M}^X_{j,2k+1} + d_{\tilde{M}^X_{j,2k+1}}\log \times \sum_{l=-L}^{L} C_{N,2l+N}\left(d_{\log \tilde{M}^X_{j,2k+1}}\exp \times(\log M^X_{j-1,k+l} - \log \tilde{M}^X_{j,2k+1})\right) \\
&= \log \tilde{M}^X_{j,2k+1} + \underbrace{d_{\tilde{M}^X_{j,2k+1}}\log \times d_{\log \tilde{M}^X_{j,2k+1}}\exp}_{=Id} \times \sum_{l=-L}^{L} C_{N,2l+N}(\log M^X_{j-1,k+l} - \log \tilde{M}^X_{j,2k+1}) \\
&= \log \tilde{M}^X_{j,2k+1} + \sum_{l=-L}^{L} C_{N,2l+N}\log M^X_{j-1,k+l} - \underbrace{\sum_{l=-L}^{L} C_{N,2l+N}}_{=1}\log \tilde{M}^X_{j,2k+1} \\
&= \sum_{l=-L}^{L} C_{N,2l+N}\log M^X_{j-1,k+l} \\
&= \sum_{l=-L}^{L} C_{N,2l+N}\log M^\gamma_{j-1,k+l} + \sum_{l=-L}^{L} C_{N,2l+N}\log M^\xi_{j-1,k+l} \quad \text{from (9)} \\
&= \log \tilde{M}^\gamma_{j,2k+1} + \log \tilde{M}^\xi_{j,2k+1} \qquad (10)
\end{aligned}$$

where we used the weighted mean of the $(j-1)$-scale neighbouring midpoints to compute the predicted point $\tilde{M}^X_{j,2k+1}$. Also, as we have seen before, the sum of the filter coefficients equals 1. The identity used in the product of the 2 differentials shows that indeed the Fréchet derivative is not needed in this context. The last two relations actually lead to the following one for the whitened wavelet coefficients:

$$\begin{aligned}
\mathcal{D}^X_{j,k} &:= 2^{-j/2}(\log M^X_{j,2k+1} - \log \tilde{M}^X_{j,2k+1}) \\
&= 2^{-j/2}(\log M^\gamma_{j,2k+1} - \log \tilde{M}^\gamma_{j,2k+1}) + 2^{-j/2}(\log M^\xi_{j,2k+1} - \log \tilde{M}^\xi_{j,2k+1}) \\
&= \mathcal{D}^\gamma_{j,k} + \mathcal{D}^\xi_{j,k} \qquad (11)
\end{aligned}$$

where the middle equality comes from the relations (9) and (10). Thus, we see that in the log-Euclidean framework we can express the differences in the coefficients between the input data and their prediction counterpart as a *non-scalar* signal-plus-noise model in the wavelets coefficients domain. Again, this is due to the log-Euclidean metric and the fact that the Riemannian logarithmic map and the classical logarithm result in a simple addition for the whitened wavelet coefficients in the tangent space at the identity. The difference with the trace-thresholding relation obtained with the affine-invariant metric is here important: instead of a scalar signal-plus-noise model, we now get a matrix-valued model. Obviously, this allows for more general thresholding procedures than in the Riemannian manifold with the affine-invariant metric. The reason for this is due to the relaxed symmetry constraints, along with the easier Euclidean computation framework, with the choice of the log-Euclidean metric.

5.2 Choice of the thresholding rule and comparison to the Riemannian metric

In (Chau 2018) - proposition 3.4.3, the thresholding rule found, considering that the data are HPD matrix-valued curves, is the following trace-thresholding:

$$\mathrm{Tr}(\mathcal{D}^X_{j,k}) = \mathrm{Tr}(\mathcal{D}^f_{j,k}) + \mathrm{Tr}(\mathcal{D}^W_{j,k})$$

where the whitened wavelet coefficients at each scale-location (j,k), obtained via the intrinsic affine-invariant wavelet transform of order $N = 2L+1 \geq 1$, are based on the observed sequence $X_{l=1,...,n} = (f^{1/2} * W)_{l=1,...,n}$ where f is a deterministic sequence and W_l is a sequence of i.i.d. Wishart matrices (with L degree of freedom and intrinsic mean equal to the identity). They found that a type of scalar signal-plus-noise model, in the wavelet domain, can be computed by using the trace of the matrix-valued coefficients.
Based on the relations found in the last section, we have obtained a matrix-valued signal-plus-noise model. We now look at possible wavelet thresholding rules for the Riemannian manifold with the log-Euclidean metric.

Choice of the thresholding rule The choice of the thresholding rule will obviously depends on the type of thresholding (hard, soft,...) and on the way of using the information inside the data related to the threshold value. Based on the relation (11), we could pick within the following options:

1. component-wise scalar scheme with $d(d+1)/2$ relations: $[\mathcal{D}^X_{j,k}]_{a,b} = [\mathcal{D}^\gamma_{j,k}]_{a,b} + [\mathcal{D}^\xi_{j,k}]_{a,b}$ with $b \geq a \in (1,...,d)$ where the matrices are supposed to have dimension $d \times d$
2. sum-of-components scalar scheme: $\sum_{a,b}[\mathcal{D}^X_{j,k}]_{a,b} = \sum_{a,b}[\mathcal{D}^\gamma_{j,k}]_{a,b} + \sum_{a,b}[\mathcal{D}^\xi_{j,k}]_{a,b}$
3. trace scalar scheme: $\mathrm{Tr}(\mathcal{D}^X_{j,k}) = \mathrm{Tr}(\mathcal{D}^\gamma_{j,k}) + \mathrm{Tr}(\mathcal{D}^\xi_{j,k})$
4. ...

Obviously, those are all interesting to investigate with different types of thresholding. Still, as we want to draw comparisons with the affine-invariant approach from (Chau 2018), we will consider the number 2 option of the preceding list. Indeed, it gives us a scalar signal-plus-noise model where we actually take more information than in the trace-thresholding (TT) approach because we make the summation on the whole elements and not only the diagonal of the matrix. We will refer to this rule as the sum-thresholding (ST). Therefore, we define $d_{j,k} = \sum_{a,b}[\mathcal{D}^X_{j,k}]_{a,b}$, where we sum up all the matrix (whitened) coefficients at each scale-location (j,k). As said before, we will also use the tree-structured scheme, described in section 2.3.3, where we will also use the typical *universal threshold*, used in classical wavelet theory (Donoho 1997) as well as in (Chau 2018). The scalar signal-plus-noise approach is particularly simple and well-known, but there are some constraints to respect before implementing it. Indeed (Jansen 2001), we have to verify that the variance of the noise is homogeneous across coefficient scales. As a first step, let's compute the expectation and variance of the relation

(11).

From (11) and the fact that we know the noise probability distribution (see section 3.1), we find that:

$$\begin{aligned}
\mathbb{E}(\mathcal{D}^X_{j,k}) &= \mathbb{E}(\mathcal{D}^\gamma_{j,k}) + \mathbb{E}(\mathcal{D}^\xi_{j,k}) \\
&= \mathcal{D}^\gamma_{j,k} + \mathbb{E}\left(2^{-j/2}(\log M^\xi_{j,2k+1} - \log \tilde{M}^\xi_{j,2k+1})\right) \\
&= \mathcal{D}^\gamma_{j,k} + 2^{-j/2}\mathbb{E}\left(\frac{1}{2^{J-j}}\sum_{l=0}^{2^{J-j}-1}\log(e^{\xi_{2k^{J-j-1}+l+1}}) - \sum_{l=-L}^{L} C_{N,2l+N}\log M^\xi_{j-1,k+l}\right) \\
&= \mathcal{D}^\gamma_{j,k} + 2^{-j/2}\left(\frac{1}{2^{J-j}}\sum_{l=0}^{2^{J-j}-1}\underbrace{\mathbb{E}(\xi_{2k^{J-j-1}+l+1})}_{=0} - \sum_{l=-L}^{L} C_{N,2l+N}\mathbb{E}(\log M^\xi_{j-1,k+l})\right) \\
&= \mathcal{D}^\gamma_{j,k} \qquad (12)
\end{aligned}$$

For the variance, we have:

$$\begin{aligned}
\mathrm{Var}(\mathcal{D}^X_{j,k}) &= 2^{-j}\mathrm{Var}(\log M^\gamma_{j,2k+1} - \log \tilde{M}^\gamma_{j,2k+1} + \log M^\xi_{j,2k+1} - \log \tilde{M}^\xi_{j,2k+1}) \\
&= 2^{-j}\mathrm{Var}(\log M^\xi_{j,2k+1} - \log \tilde{M}^\xi_{j,2k+1}) \quad \text{(the constant term vanishes under Var}(\cdot)) \\
&= 2^{-j}\mathrm{Var}\left(\log M^\xi_{j,2k+1} - \sum_{l=-L}^{L} C_{N,2l+N}\log M^\xi_{j-1,k+l}\right) \\
&= 2^{-j}\mathrm{Var}\left(\log M^\xi_{j,2k+1} - \overbrace{C_{N,N}}^{=1}\log M^\xi_{j-1,k} - \sum_{-L\le l\le L, l\neq 0} C_{N,2l+N}\log M^\xi_{j-1,k+l}\right) \qquad (13)
\end{aligned}$$

Now, we know that the midpoints, at a specific scale j, are i.i.d. with each other for all k. Also, we can compute the variance of $\log M^\xi_{j-1,k}$ using the relation (6):

$$\begin{aligned}
\mathrm{Var}(\log M^\xi_{j-1,k}) &= \mathrm{Var}\left(\frac{1}{2}\log M^\xi_{j,2k} + \frac{1}{2}\log M^\xi_{j,2k+1}\right) \\
&= \frac{1}{2}\mathrm{Var}(\log M^\xi_{j,0}) = ... \\
&= \frac{2^{-1}}{2^{J-j}}\mathrm{Var}(\log M^\xi_{J,0}) = \frac{2^{-1}}{2^{J-j}}\mathrm{Var}(\xi_1) \qquad (14)
\end{aligned}$$

Hence, based on (14), the relation (13) becomes:

$$\begin{aligned}
\mathrm{Var}(\mathcal{D}^X_{j,k}) &= 2^{-j-1}\mathrm{Var}(\log M^\xi_{j,2k}) + 2^{-j}\left(\sum_{-L\le l\le L} C^2_{N,2l+N} - 1\right)\mathrm{Var}(\log M^\xi_{j-1,k+l}) \\
&= 2^{-j-1}\sum_{-L\le l\le l} C^2_{N,2l+N}\mathrm{Var}(\log M^\xi_{j,0}) \\
&= \frac{2^{-j-1}}{2^{J-j}}\sum_{-L\le l\le l} C^2_{N,2l+N}\mathrm{Var}(\xi_1) \\
&= 2^{-1-J}\sum_{-L\le l\le l} C^2_{N,2l+N}\mathrm{Var}(\xi_1) \qquad (15)
\end{aligned}$$

which is a well constant variance across all scales (independent of j). In the case of our option, the ST rule, we have:

$$\sum_{a,b}[\mathcal{D}^X_{j,k}]_{a,b} = \sum_{a,b}[\mathcal{D}^\gamma_{j,k}]_{a,b} + \sum_{a,b}[\mathcal{D}^\xi_{j,k}]_{a,b}$$

with

$$\mathbb{E}\left(\sum_{a,b}[\mathcal{D}^X_{j,k}]_{a,b}\right) = \sum_{a,b}[\mathcal{D}^X_{j,k}]_{a,b} \tag{16}$$

and

$$\mathrm{Var}(\sum_{a,b}[\mathcal{D}^X_{j,k}]_{a,b}) = \sum_{a,b}\mathrm{Var}([\mathcal{D}^X_{j,k}])_{a,b} = 2^{-1-J}\sum_{-L\leq l\leq l} C^2_{N,2l+N}\sum_{a,b}[\mathrm{Var}(\xi_1)]_{a,b} \tag{17}$$

because of the assumption we made in the data model, that is independence between the component-wise variance of $\mathrm{Var}(\xi_1)$. For sufficiently large n, the scalar coefficients $d_{j,k} = \sum_{a,b}[\mathcal{D}^X_{j,k}]_{a,b}$ are approximately normally distributed at reasonably coarse scales j (Donoho 1997), for which we now have the relations (16) and (17). In this context, the universal threshold is written as $\lambda \sim \sigma_w\sqrt{2\log(N)}$, with σ^2_w being the noise variance determined either via the variance of $d_{j,k}$ or from the data itself. Another method to determine an appropriate smoothing parameter λ, directly from the data, is by the use of cross-validation (G.P. Nason 1996). As before, the tree-structured scheme, described in 2.3.3, is applied on the thresholded coefficients $d'_{j,k} \in \{d_{j,k}, 0\}$.

Other thresholding rule in the affine-invariant metric framework

Let's replace ourselves in the context of nonparametric estimation of a HPD matrix-valued curve in the Riemannian manifold with the affine-invariant metric. Upon further investigation, we notice that the following choice of thresholding rule could be suitable from an equivariance standpoint:

$$g(\det(\mathcal{D}^X_{j,k}))$$

We can verify that, upon applying a general congruence transformation $A * X$, we find:

$$\begin{aligned}\det(\mathcal{D}^{X,A}_{j,k}) &= \det\left((\tilde{M}^{X,A}_{j,k})^{-1/2} * D^{X,A}_{j,k}\right) = \det\left((A^*\tilde{M}^X_{j,k}A)^{-1}\right)\det(A^\dagger D^X_{j,k}A)\\ &= \det\left((\tilde{M}^X_{j,k})^{-1}\right)\det(D^X_{j,k}) = \det\left((\tilde{M}^X_{j,k})^{-1/2}D^X_{j,k}(\tilde{M}^X_{j,k})^{-1/2})\right)\\ &= \det\left((\tilde{M}^X_{j,k})^{-1/2} * D^X_{j,k}\right) = \det(\mathcal{D}^X_{j,k})\end{aligned}$$

using classical properties of the determinants, and the Hermitian property of matrices. The possible thresholding rule above is thus preserving the general linear congruence equivariance. Naturally, the use of the determinant thresholding will depend on the link between the signal and noise, ie. the data model, before being of any use. As we have seen, we could use the eigendecomposition method of a SPD matrix and write

$$\mathcal{D}_{j,k} = Q_{j,k}\Lambda_{j,k}Q^T_{j,k} \tag{18}$$

where the orthogonal matrices $Q_{j,k}$, containing the eigenvectors, and the diagonal matrices $\Lambda_{j,k}$, containing the eigenvalues, all depend on the levels j, k. Taking the trace of (18), as in the TT rule, we

can interpret the trace of the observed whitened coefficients as an addition of the sum of signal-related eigenvalues and the sum of noise-related eigenvalues. Taking the determinant of (18), we get the product of the observed eigenvalues. A model that would fit this setting would make the following connection between the signal and noise in the wavelet domain:

$$\mathcal{D}_{j,k}^{X} = \mathcal{D}_{j,k}^{\text{signal}} \times \mathcal{D}_{j,k}^{\text{noise}}$$

Further research on the latter description would be interesting, especially in the choice of a suitable data model.

6 Comparison between the affine-invariant Riemannian metric and the log-Euclidean metric by means of simulated data

In this section, we present the results of our nonparameric estimation via wavelet thresholding based on simulated SPD matrix-valued curves. The underlying goal of our study is 2-fold. We want to draw:

- comparison between the sum-thresholding rule (noted ST) and the trace-thresholding (noted TT)
- comparison between the log-Euclidean and the affine-invariant metrics within the TT rule

In order to make the comparisons between the different metrics and threshdolding rules, we will consider simple jump functions as the entries of our SPD matrix-valued curves. The idea is to see how the efficiency of the different approaches evolve with increasing curve complexity. We will therefore add more irregularity on the different components. The next criteria are used both as a parameter setting and as a mean of comparisons between the 3 estimation methods:

- **Graphical similarity** between the estimates and the original signal
- **Mean squared error minimization** between the estimates and the signal, averaged through a number of simulations
- **Computation time** of a given estimation procedure, averaged through a number of simulations

The first criterium serves to determine the level of non-linear thresholding to perform in order to obtain an estimate that have a good balance between noise-removal and over-fitting. Also, it permits to compare graphically the results obtained. The second and the third allow to range the efficiency of the procedures both in terms of accuracy and computation time.

Non-linear thresholding As previously described, we use tree-structured thresholding for the 3 approaches. The imposed tree-structure is able to capture jump/peak discontinuities in the original signal, as wavelet coefficients of singularities typically persist across wavelet scales at the location of the singularity. Conversely, isolated large wavelet coefficients, usually arising from noise and not signal, are set to zero, since they are not part of the tree. In order to keep a given coefficient, one has to meet the following condition:

$$|d_{j,k}| > \lambda = \alpha\sigma_w\sqrt{2\log N}$$

where λ is our smoothing parameter. This parameter contains N, the number of computed coefficients, the noise variance σ_w^2 and a fine-tuning parameter α which serves to determine the degree of nonlinear thresholding to perform. In the present case, we will determine this parameter *by-hand* because we did not investigate into cross-validation methods (G.P. Nason 1996) to compute it more precisely. This will have an important impact in the interpretation of our results.

The noise variance σ_w^2 is calculated from the maximum between (17) and the variance evaluated from the data. In the ST case, σ_w will written as:

$$\sigma_w = \max\left\{\text{MAD}, \sqrt{2^{-J} d \times \sigma \sum_{-L\leq l\leq L} C_{N,2l+N}^2}\right\}$$

where $d = 2$ (row-dimension of the data), N is the refinement order, σ is the component-wise variance of the noise term in the data model (8) and MAD refers to the Median Absolute Deviation obtained from the observed $d_{j,k}$. Before going on with simulations, we need to adapt the TT rule to our data model. In both approaches, we will disregard a coefficient in the case where:

- for TT: $|\text{Tr}(\mathcal{D}_{j,k})| \leq \alpha_{TT}\sqrt{2\log(N)}\sigma_w^{TT}$
- for ST: $|\sum_{a,b}[\mathcal{D}_{j,k}]_{a,b}| \leq \alpha_{ST}\sqrt{2\log(N)}\sigma_w^{ST}$

where both α-parameters control the level non-linear thresholding and the sub/lower-script refers to the rule used. Both rules relies on sum of elements of the withened coefficients. As we know the component-wise noise variance, σ, and the fact that the TT rule differs from the ST rule by a summation of 2 components instead of 4 (for $d = 2$), we actually have a link between σ_w^{TT} and σ_w^{ST}.

6.1 Simulation setup and parameters

Next, we fix the parameters and the functions (ie. original signal, without noise) used for our data model (8), with the component-wise variance σ^2, and which are then denoised via the thresholding methods based on the rules and metrics described above.

Tests functions Below, we list the functions, namely curves of SPD matrices, that will serve us as benchmark (background signal) to perform wavelet non-linear thresholding. Those curves all composed of 2×2 SPD matrices, with a parameter $t \in [0, 1)$.

1. $\gamma_1(t) = \begin{pmatrix} (20-10t)_{\mathbb{1}_{t\leq 1/3}} + (22-10t)_{\mathbb{1}_{t>1/3}} & 0 \\ 0 & (20-10t)_{\mathbb{1}_{t\leq 1/3}} + (22-10t)_{\mathbb{1}_{t>1/3}} \end{pmatrix}$

2. $\gamma_2(t) = \begin{pmatrix} (20-10t)_{\mathbb{1}_{t\leq 1/3}} + (22-10t)_{\mathbb{1}_{t>1/3}} & (10-5t) \\ (10-5t) & (20-10t)_{\mathbb{1}_{t\leq 1/3}} + (22-10t)_{\mathbb{1}_{t>1/3}} \end{pmatrix}$

3. $\gamma_3(t) = \begin{pmatrix} (20-10t)_{\mathbb{1}_{t\leq 1/3}} + (22-10t)_{\mathbb{1}_{t>1/3}} & (11-10t)_{\mathbb{1}_{t\leq 1/2}} + (15-10t)_{\mathbb{1}_{t>1/2}} \\ (11-10t)_{\mathbb{1}_{t\leq 1/2}} + (15-10t)_{\mathbb{1}_{t>1/2}} & (20-10t)_{\mathbb{1}_{t\leq 1/3}} + (22-10t)_{\mathbb{1}_{t>1/3}} \end{pmatrix}$

4. $\gamma_4(t) = \begin{pmatrix} (20-10t)_{\mathbb{1}_{t\leq 1/3}} + (22-10t)_{\mathbb{1}_{t>1/3}} & (11-10t)_{\mathbb{1}_{t\leq 1/2}} + (15-10t)_{\mathbb{1}_{t>1/2}} \\ (11-10t)_{\mathbb{1}_{t\leq 1/2}} + (15-10t)_{\mathbb{1}_{t>1/2}} & (27-14t)_{\mathbb{1}_{t\leq 2/3}} + (20-14t)_{\mathbb{1}_{t>2/3}} \end{pmatrix}$

5. $\gamma_5(t) = \begin{pmatrix} (20-10t)_{\mathbb{1}_{t\leq 1/3}} + (22-10t)_{\mathbb{1}_{t>1/3}} & (1-10t)_{\mathbb{1}_{t\leq 1/2}} + (5-10t)_{\mathbb{1}_{t>1/2}} \\ (1-10t)_{\mathbb{1}_{t\leq 1/2}} + (5-10t)_{\mathbb{1}_{t>1/2}} & (27-14t)_{\mathbb{1}_{t\leq 2/3}} + (20-14t)_{\mathbb{1}_{t>2/3}} \end{pmatrix}$

The continuous functions written above will be sampled on a discrete grid of equally distant steps, ie. $\gamma(i/n)_{i=0,..,n-1}$ with n being the total number of observations. The parameters of our simulations are the following:

- number of observations: $n = 2^J$
- sampling scale: $J = 10$
- refinement order: $N = 5$
- $\alpha = 0.65$
- $\sigma = 0.1$

Those will be fixed for all the simulations. Computationally speaking, we had to adapt the code from (Chau 2020) R-package `pdSpecEst`. The overall scheme to apply are the following steps when one wants to perform nonlinear wavelet thresholding based on the ST/TT rule:

1 Compute the SPD matrix-valued curve, with the points γ.

2 Compute he noisy version of the original signal via our model: $X(i/n) = \exp(\log \gamma(i/n) + \xi(i/n))$ where the variance of each component of ξ is σ^2.

3 Perform a forward intrinsic AI wavelet transform (with `WavTransf1D`) - get to the wavelet domain

4 Apply the tree-structured thresholding of the wavelet coefficients (with `spdCART`/`TTpdCART`)

5 Perform the inverse intrinsic AI wavelet transform (with `InvWavTransf1D`) - get back in the data domain

Similarly, we also adapted the code for the TT rule to our data model, as well as to account for the choice of α made above.

6.2 Results, observations and interpretation

Below, we show the different results we found, with observations. We will interpret the whole at the end of the section. As we said, the component-wise standard deviation is $\sigma = 0.1$ and, given the link between σ_w^{TT} and σ_w^{ST}, we set the choice of α based on a compromise between the *goodness-of-fit* and overfitting in the graphical tests for both ST and TT rules. We found that the nonlinear fine-tuning parameter could be $\alpha = 0.65$.

Case 1 - Simulation from curve γ_1 This curve presents the same jump discontinuity on the diagonal entries of each SPD matrices while the off-diagonal elements are set to 0. The following figure represents the component-wise curve of SPD matrices:

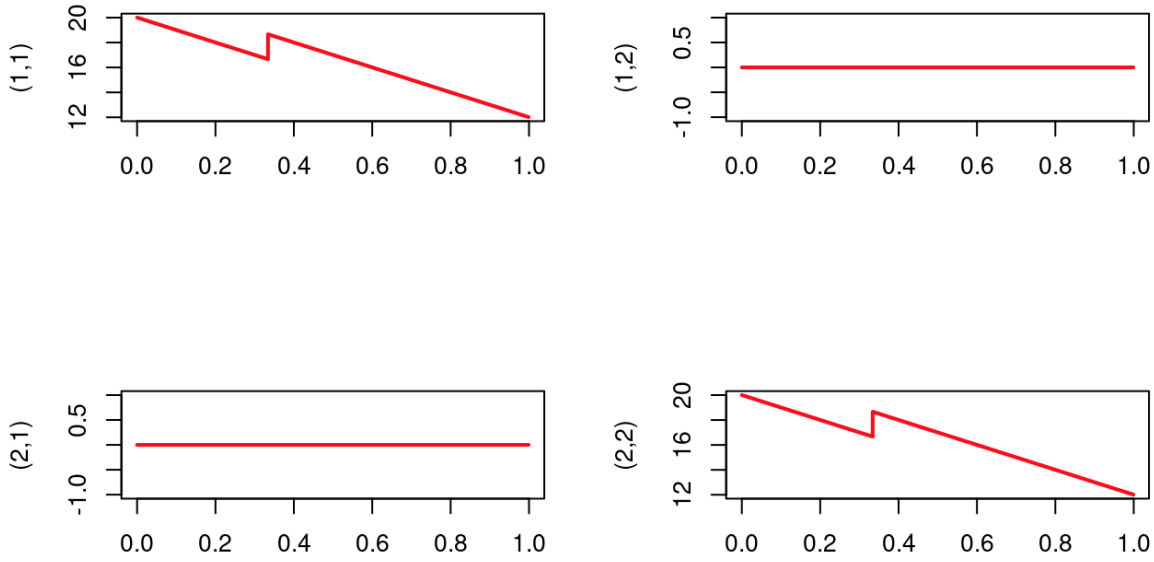
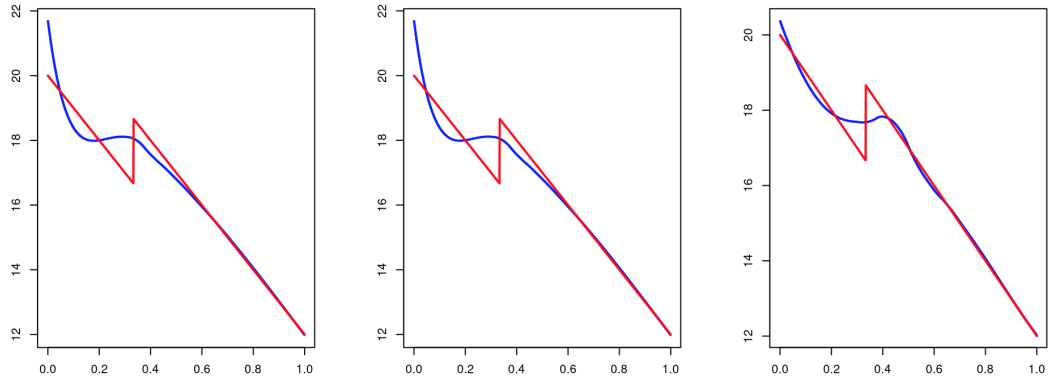
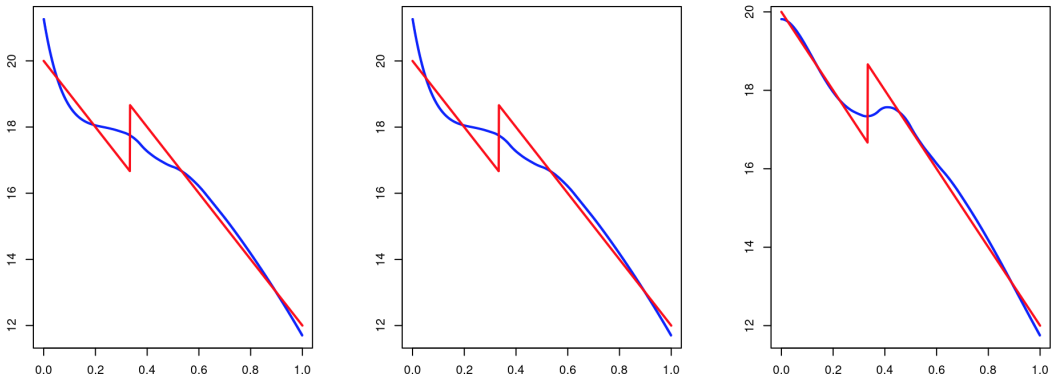


Figure 5: γ_1 curve

With the parameters listed above, we obtain the following graphical representation, for a fixed simulation of the signal-plus-noise model, of the estimated signal via the TT rule (under both the log-Euclidean and the affine-invariant metrics) and via the ST rule under the log-Euclidean metric.



(a) (1,1)-component of the original curve and its estimate.



(b) (2,2)-component of the original curve and its estimate.

Figure 6: For each picture, left to right: TT rule with log-Euclidean metric, TT rule with affine-invariant metric, ST rule with log-Euclidean metric

	Average MSE after 100 simulations	Average computation time after 1000 simulations
TT with log-Euclidean metric	0.001092	0.009921
TT with affine-invariant metric	0.001115	0.04081
ST with log-Euclidean metric	0.001276	0.007547

Table 3: Average MSE and average computation time after 100 and 1000 simulations, respectively.

Observations: There is no big graphical differences between the three approaches, for both components, eventhough the ST rule seems to fit better overall. The 3 approaches seems to be equivalent in terms of the mean squared error (MSE), the TT rule with the log-Euclidean metric being the most accurate. Computation time is on average almost 5 times slower with the affine-invariant metric, compared to the log-Euclidean one's.

Case 2 - Simulation from curve γ_2 This curve presents the same jump discontinuity on the diagonal entries of each SPD matrices while the off-diagonal elements are set to the same linear functions. The following figure represents the component-wise curve of SPD matrices:

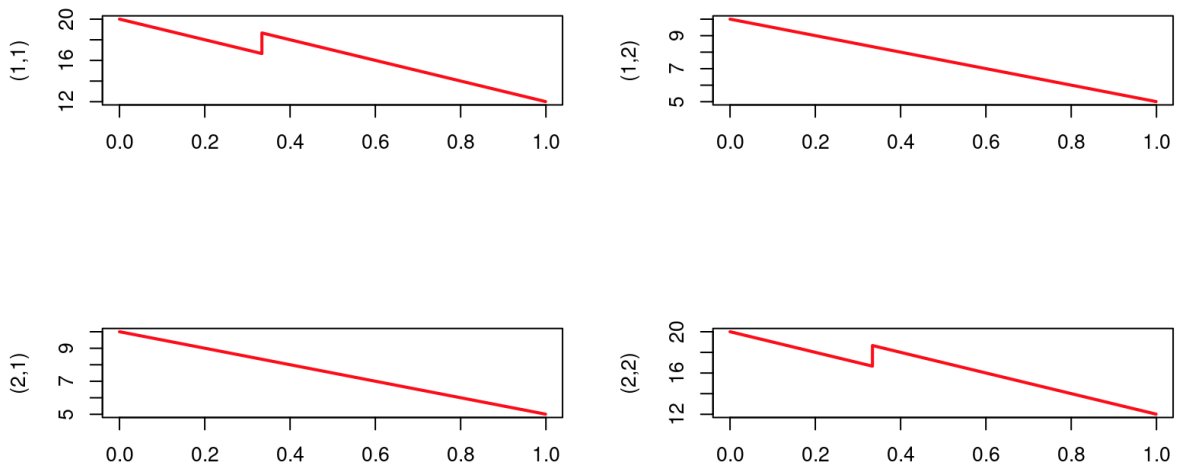


Figure 7: γ_2 curve

Again, we find a typical graphical representation along with the table contained average MSE and average computation time values.

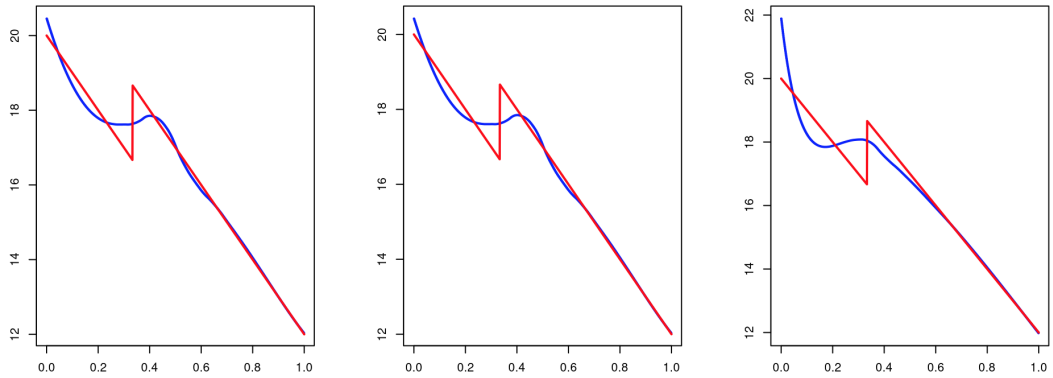


Figure 8: (1,1)-component. From left to right: TT rule with log-Euclidean metric, TT rule with affine-invariant metric, ST rule with log-Euclidean metric

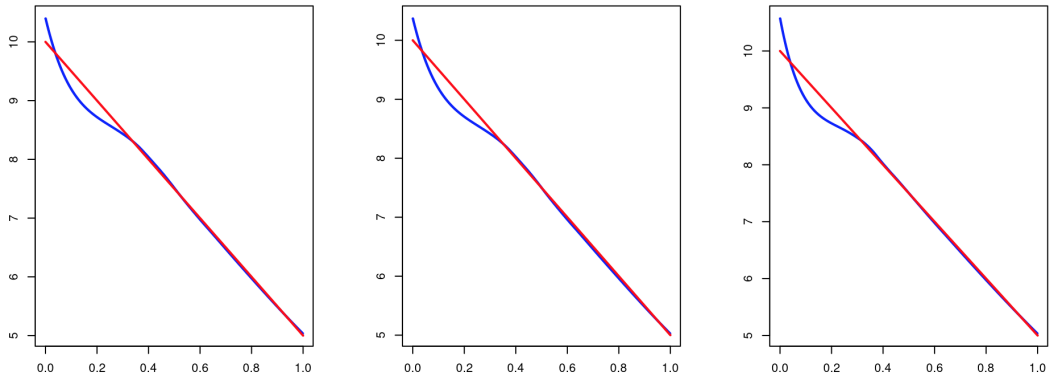


Figure 9: (1,2)-component. From left to right: TT rule with log-Euclidean metric, TT rule with affine-invariant metric, ST rule with log-Euclidean metric

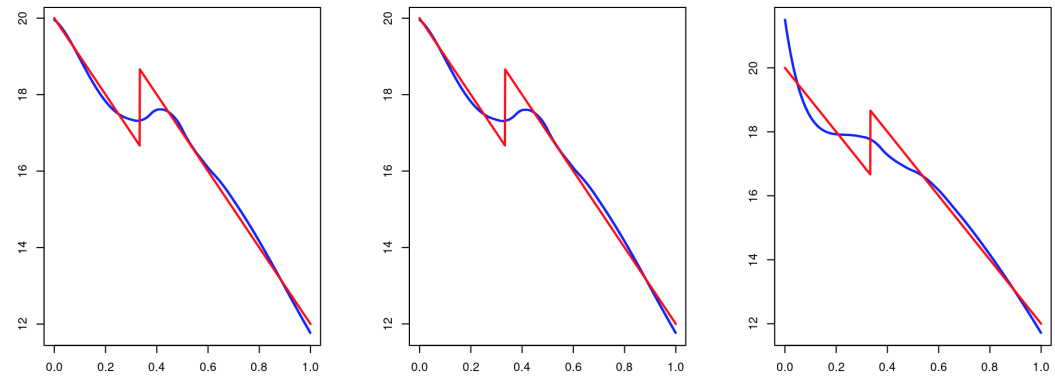


Figure 10: (2,2)-component. From left to right: TT rule with log-Euclidean metric, TT rule with affine-invariant metric, ST rule with log-Euclidean metric

	Average MSE after 100 simulations	Average computation time after 1000 simulations
TT with log-Euclidean metric	0.001242	0.009847
TT with affine-invariant metric	0.001187	0.04187
ST with log-Euclidean metric	0.002488	0.007649

Table 4: Average MSE and average computation time after 100 and 1000 simulations, respectively.

Observations: The ST approach is globally less performant in terms of graphical display than the two others (which are equivalent to each other). Graphically and in terms of the MSE, the log-Euclidean metric and affine-invariant metric, under the TT rule, seems equivalent. The ST rule is about 2.1 times bigger, in terms of the MSE, than the TT rules. Computation time is again, on average, almost 5 times slower with the affine-invariant metric, compared to the log-Euclidean one's.

Case 3 - Simulation from curve γ_3 This curve presents the same jump discontinuity on the diagonal entries of each SPD matrices while the off-diagonal elements present another jump (the same

for each component) located at a different position. The following figure represents the component-wise curve of SPD matrices:

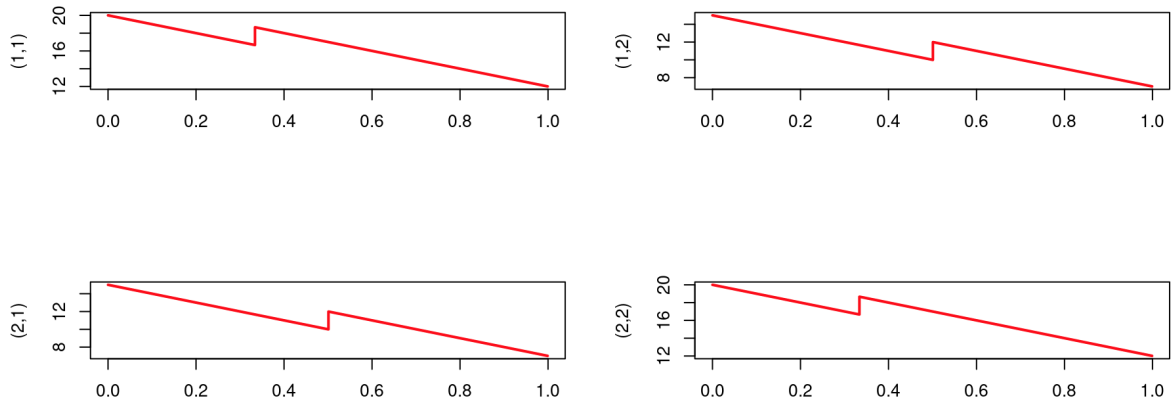


Figure 11: γ_3 curve

Next, we give a typical graphical representation along with the table contained average MSE and average computation time values.

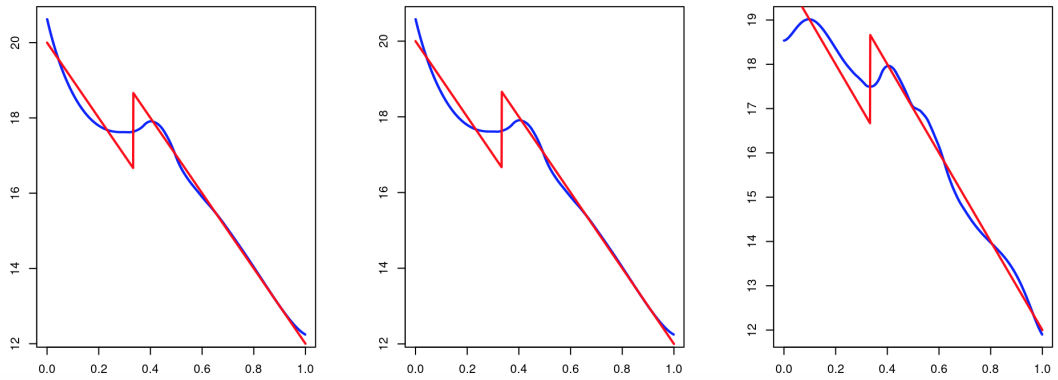


Figure 12: (1,1)-component. From left to right: TT rule with log-Euclidean metric, TT rule with affine-invariant metric, ST rule with log-Euclidean metric

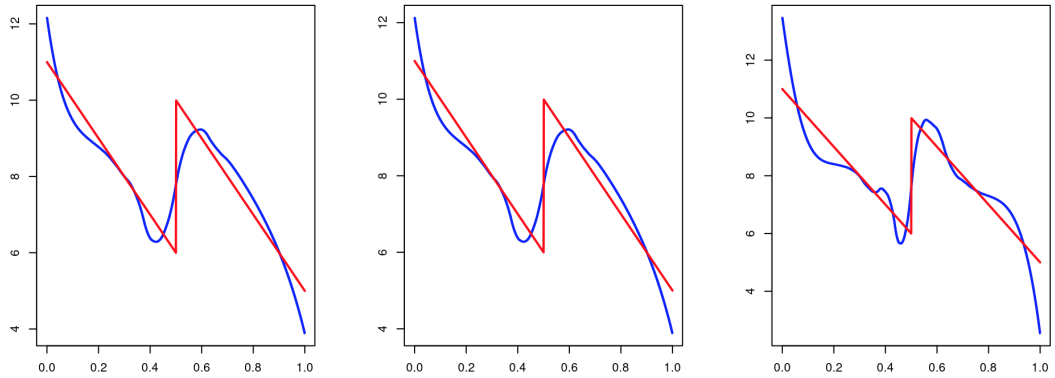


Figure 13: (1,2)-component. From left to right: TT rule with log-Euclidean metric, TT rule with affine-invariant metric, ST rule with log-Euclidean metric

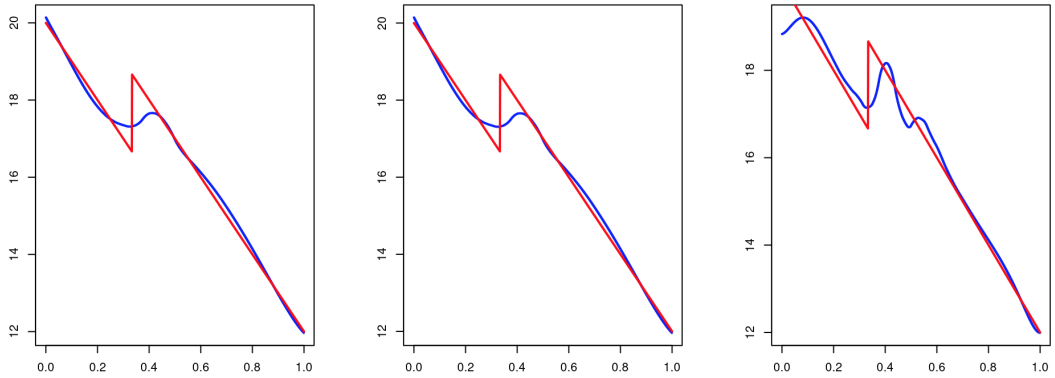


Figure 14: (2,2)-component. From left to right: TT rule with log-Euclidean metric, TT rule with affine-invariant metric, ST rule with log-Euclidean metric

	Average MSE after 100 simulations	Average computation time after 1000 simulations
TT with log-Euclidean metric	0.004620	0.009855
TT with affine-invariant metric	0.004643	0.04091
ST with log-Euclidean metric	0.01063	0.007685

Table 5: Average MSE and average computation time after 100 and 1000 simulations, respectively.

Observations: There are notable graphical differences between the TT and ST approaches, the latter being less performant globally. Graphically and in terms of the MSE, the log-Euclidean metric and affine-invariant metric, under the TT rule, seems equivalent. The ST rule is about 2.3 times bigger, in terms of the MSE, than the TT rules. Computation time is again, on average, almost 5 times slower with the affine-invariant metric, compared to the log-Euclidean one's.

Case 4 - Simulation from curve γ_4 This curve presents different (height and locations) jumps discontinuity on the diagonal entries of each SPD matrices while the off-diagonal elements are having the same jump discontinuity at the same location. The following figure represents the component-wise curve of SPD matrices:

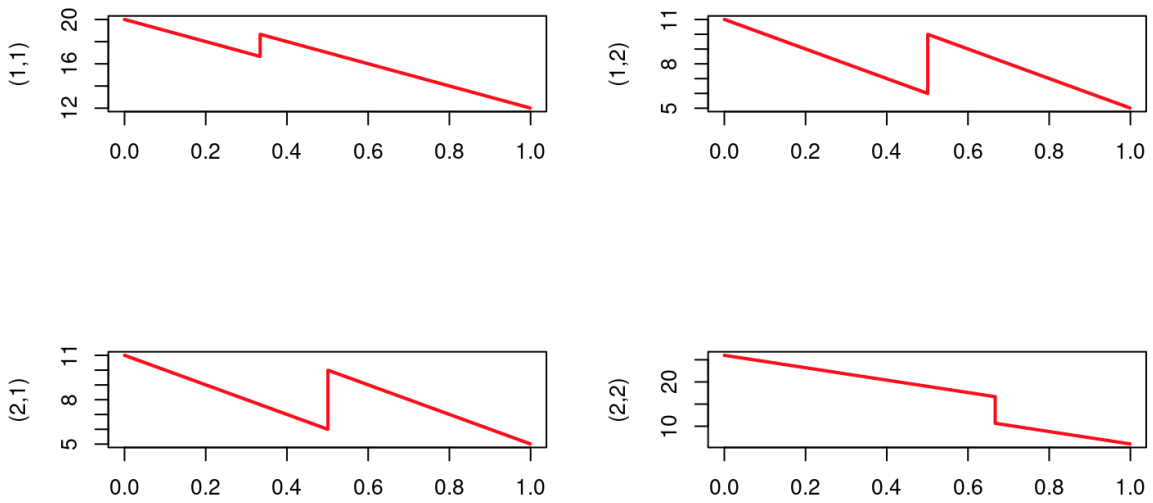


Figure 15: γ_4 curve

Next, we give a typical graphical representation along with the table contained average MSE and average computation time values.

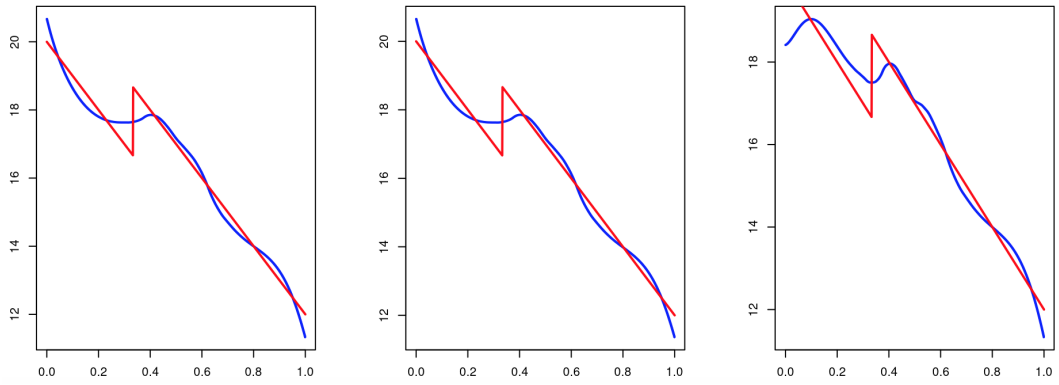


Figure 16: (1,1)-component. From left to right: TT rule with log-Euclidean metric, TT rule with affine-invariant metric, ST rule with log-Euclidean metric

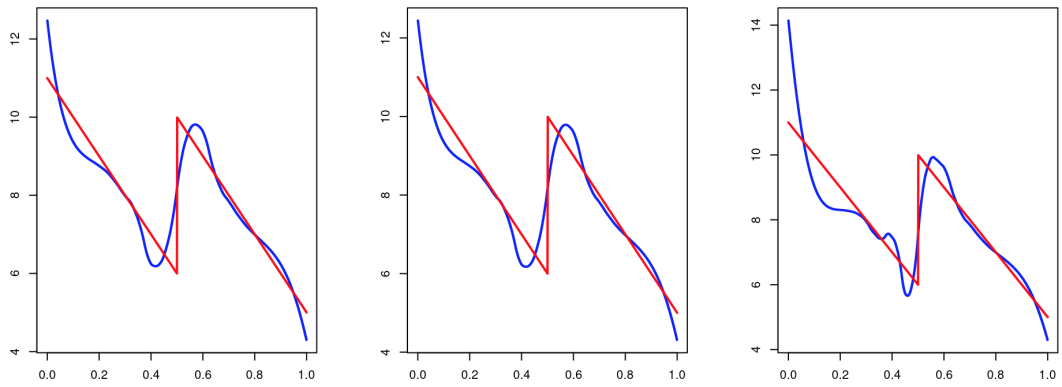


Figure 17: (1,2)-component. From left to right: TT rule with log-Euclidean metric, TT rule with affine-invariant metric, ST rule with log-Euclidean metric

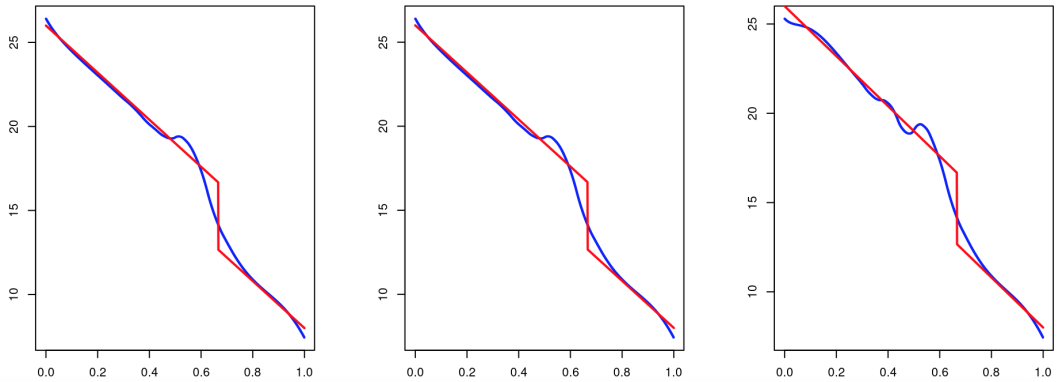


Figure 18: (2,2)-component. From left to right: TT rule with log-Euclidean metric, TT rule with affine-invariant metric, ST rule with log-Euclidean metric

	Average MSE after 100 simulations	Average computation time after 1000 simulations
TT with log-Euclidean metric	0.003831	0.009864
TT with affine-invariant metric	0.003871	0.04136
ST with log-Euclidean metric	0.009710	0.007761

Table 6: Average MSE and average computation time after 100 and 1000 simulations, respectively.

Observations: Overall, the shape of the estimates for each component seems to be globally equivalent, with a slight advantage to the TT rules. Graphically and in terms of the MSE, the log-Euclidean metric and affine-invariant metric, under the TT rule, seems equivalent. The ST rule is about 2.5 times bigger, in terms of the MSE, than the TT rules. Computation time is again, on average, almost 5 times slower with the affine-invariant metric, compared to the log-Euclidean one's.

Case 5 - Simulation from curve γ_5 This curve presents different (height and locations) jumps discontinuity on the diagonal entries of each SPD matrices while the off-diagonal elements are having the same jump discontinuity at the same location. The following figure represents the component-wise curve of SPD matrices:

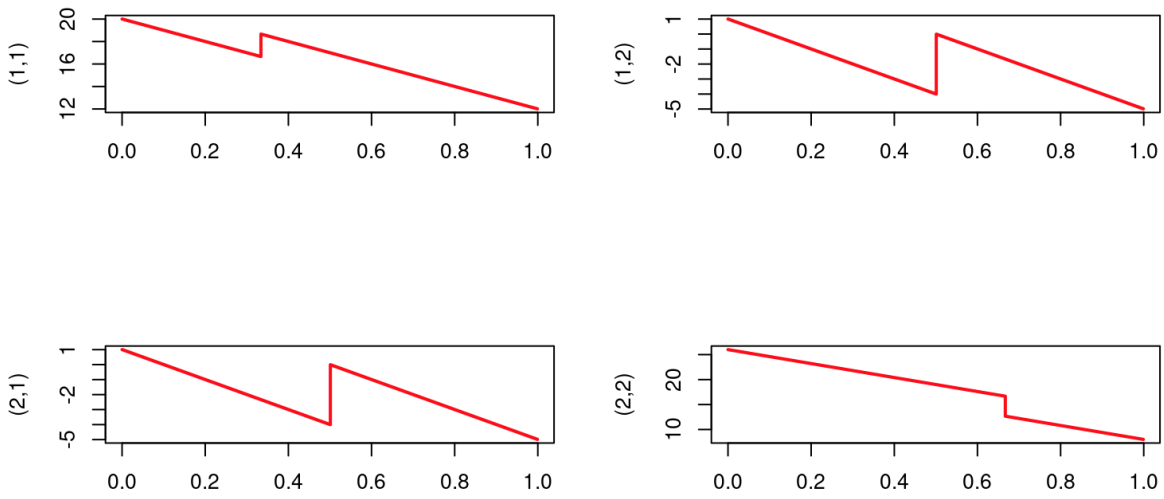


Figure 19: γ_4 curve

Next, we give a typical graphical representation along with the table contained average MSE and average computation time values.

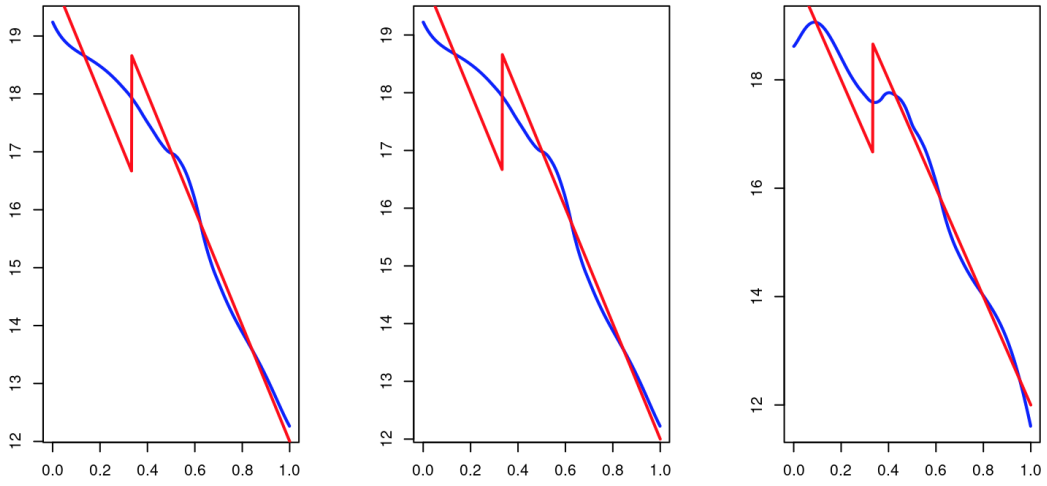


Figure 20: (1,1)-component. From left to right: TT rule with log-Euclidean metric, TT rule with affine-invariant metric, ST rule with log-Euclidean metric

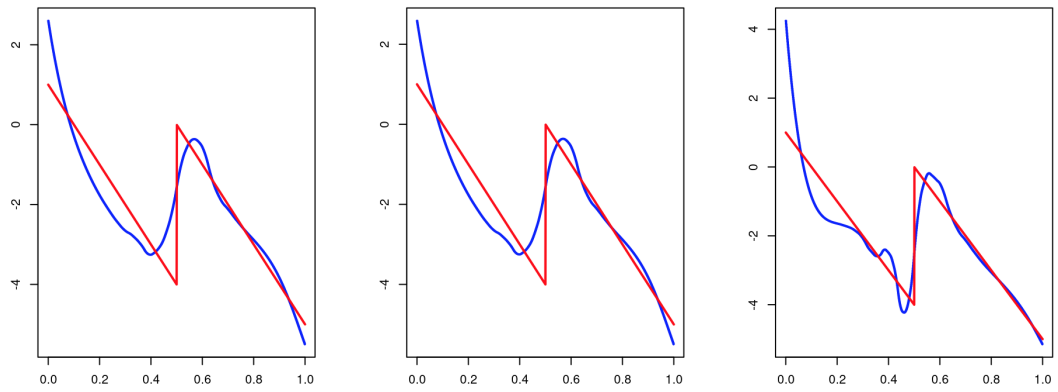


Figure 21: (1,2)-component. From left to right: TT rule with log-Euclidean metric, TT rule with affine-invariant metric, ST rule with log-Euclidean metric

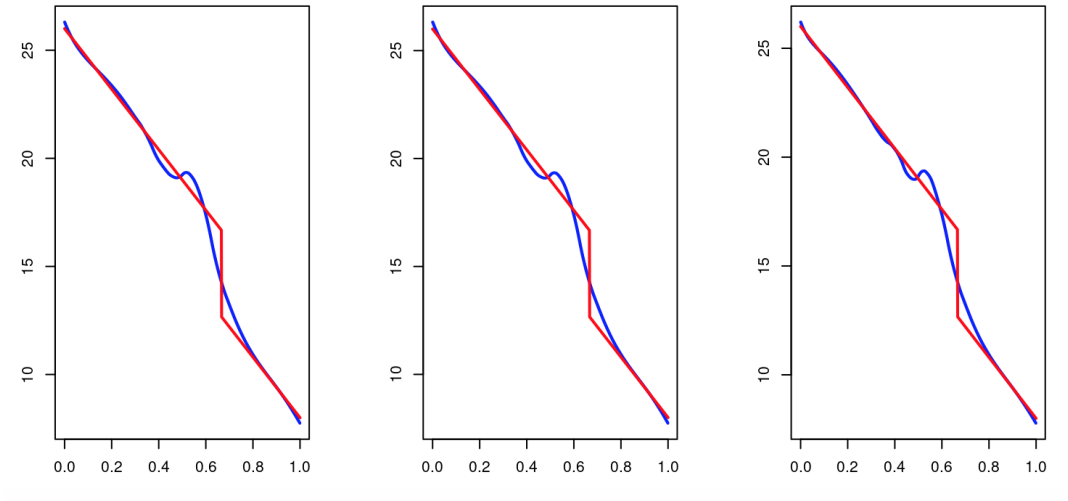


Figure 22: (2,2)-component. From left to right: TT rule with log-Euclidean metric, TT rule with affine-invariant metric, ST rule with log-Euclidean metric

	Average MSE after 100 simulations	Average computation time after 1000 simulations
TT with log-Euclidean metric	0.004992	0.01015
TT with affine-invariant metric	0.004764	0.04078
ST with log-Euclidean metric	0.03746	0.007688

Table 7: Average MSE and average computation time after 100 and 1000 simulations, respectively.

Observations: Overall, the shape of the estimates for each component seems slightly better with the ST rule, compared to the TT rules. Graphically and in terms of the MSE, the log-Euclidean metric and affine-invariant metric, under the TT rule, seems equivalent. The ST rule is about 25% smaller, in terms of the MSE, than the TT rules. Computation time is again, on average, almost 5 times slower

with the affine-invariant metric, compared to the log-Euclidean one's.

Interpretation of the results In all simulations, we clearly see that using the ST rule with the log-Euclidean metric is almost 5 times faster than the TT rule with the affine-invariant metric. Also, the ST rule is always faster than TT rule, whatever the metric being used.
In terms of accuracy, the computations suggest that, when the matrices are almost purely diagonal, then there isn't a real difference between the 3 approaches. However, as the complexity of the curves is increased, the trace-thresholding method with the affine-invariant or log-Euclidean metric are the most efficient. The two latter are globally equivalent in terms of graphical display or based on the MSE computed. This is true for all the cases until the treatment of curve γ_5, which clearly favors the ST rule in terms of the MSE. The reason for this discrepancy in our results is related to the choice of the simulated functions we considered. For the first 4 cases, we have treated SPD matrix-valued curves where each component of the matrices are almost always positive, even when noise is added. Explicitely, this means that the SPD matrices we treat in those cases have the following form:

$$A = \begin{pmatrix} a & b \\ b & d \end{pmatrix}, \quad a, b, c, d \in \mathbb{R}$$

where $a, b, c > 0$ for (most) of the last 4 cases. While the SPD constraints are: $a+d > 0$ and $ad-b^2 > 0$, we see that the first is then automatically met. For the ST and TT rules, we have

$$d_{jk}^{TT} = a + d, \quad d_{j,k}^{ST} = a + 2b + d$$

Thus, we see that the value will always be bigger for the ST rule (for the same whitened wavelet coefficient matrices). Hence, a lot of coefficients are kept in the ST rule, where they will be disregarded in the TT approach, which can be seen in some of the graphical representation where the estimates under the TT rule tend to be closer to the original signal that the estimates under the ST rule. With the fifth case, we do not have that problem and the ST rule actually takes the advantage.
Overall, we need more representative SPD curves/functions to have an accurate benchmark in order to test the different approaches. Also, there is a clear need to find the parameters α in a typical data-adaptive fashion, such as cross-validation, in order to have a more senseful ground for efficiency comparisons. Still, we find that, computationally speaking, the ST rule with the log-Euclidean metric is the best option regarding time-efficiency estimation methods.

7 Conclusions and perspectives

In this work, we considered simple signal-plus-noise models within non-Euclidean manifolds equipped with either the log-Euclidean or the affine-invariant metric. The main goal was to work on the set of SPD matrices seen as a Riemannian manifold with the log-Euclidean metric and perform nonparametric estimation using wavelet thresholding methods. With this metric, we transform the space of SPD matrices in a complete metric space and we can keep the unitary congruence invariance property, but not the general linear congruence invariance, that is found with the affine-invariant metric. Also, we *flatten* the curvature of the manifold leading to Euclidean-based calculations and we avoid the swelling effect that is very problematic in DTI. This simplifies greatly the mathematics tools we needed, especially due to the log-mapping. We adapted the average-interpolation wavelet transform of (Chau 2018) to the setting of SPD matrix-valued curves whose model is a typical signal-plus-noise and found an ensemble of thresholding rules in the log-Euclidean manifold, the generic one being a non-scalar signal-plus-noise in the wavelet coefficient domain. This generic estimator keeps the unitary congruence equivariance and allows for component-wise thresholding.
With the choice of the sum-thresholding rule, we performed denoising of simulated SPD curves, based on our data model, using the same tree-structured thresholding scheme as in (Chau 2018) where they used a trace-thresholding rule with the affine-invariant metric. The estimator found in the latter approach is actually equivariant under general linear congruence because of the use of the trace operator. Compared to our generic rule, the framework of wavelet thresholding with the affine-invariant metric actually shares the same level of equivariance, that is under orthogonal congruence transformations. Simulations results showed that, in all cases, our specific sum-thresholding rule does provide faster computation times without degrading the resulted estimator's accuracy.
Among the other contributions of this thesis is the introduction of a possible data generating process using the spectral decomposition of SPD matrices. With a simple example, we show that this approach is actually equivalent to a typical signal-plus-noise model. We also proposed another rule for the Riemannian manifold with the affine-invariant metric, using the determinant, that shares the same equivariance property as the trace-thresholding rule.

As we stated earlier, in the context of the sum-thresholding approach, it is needed to investigate further in the search of more overall representative SPD curves as simulation setup. Also, it would be needed to estimate the nonlinear parameter α, for example directly from the data, using cross-validation methods. Using geodesic interpolation and the log-Euclidean distance, we could implement the method used in (G.P. Nason 1996). More generally, it would definitely be interesting to perform nonparametric estimation using the generic rule, ie. the matrix-based signal-plus-noise model in the wavelet domain, found with the log-Euclidean metric by using component-wise thresholding. Another area of research would be to develop further the data model using the spectral representation of SPD matrices. Especially, it would be interesting to extend the model to account for noise impacting both the eigenvalues and the eigenvectors, which makes sense because they both describe the "shape" of a

SPD matrix. Lastly, looking for a data model that would be suitable for a potential thresholding rule using the determinant in the affine-invariant framework could be a relevant alternative.

8 Appendix - R code

Overall code used for the simulations.

```r

### Tests functions ###

f.3 <- function(t,p){
    z <- (10 -5*t)
  }
  return(z)
}
f.4 <-function(t,p){
  z <-(20-10*t)*as.numeric(t<=1/3)+ (22-10*t)*as.numeric(t>1/3)
  return(z)
}

f.4bis <-function(t,p){
  z <-(20-10*t)*as.numeric(t<=1/3)+ (24-10*t)*as.numeric(t>1/3)
  return(z)
}

f.5 <-function(t,p){
  z <-(11-10*t)*as.numeric(t<=1/2)+ (15-10*t)*as.numeric(t>1/2)
  return(z)
}

f.6 <-function(t,p){
  z <-(20-14*t)*as.numeric(t<=2/3)+ (27-14*t)*as.numeric(t>2/3)
  return(z)
}

f.7 <- function(t,p){
  z <-(1-10*t)*as.numeric(t<=1/2)+ (5-10*t)*as.numeric(t>1/2)
  return(z)
}

### Parameters of the simulations
J = 10  # sampling scale
N = 5    # refinement order
sigma <- array(c(0.1,  0.1, 0.1, 0.1,
                 0.1 ,  0.1, 0.1, 0.1,
                 0.1,  0.1, 0.1, 0.1,
                 0.1,  0.1, 0.1),
               dim=c(5,3))

sigmap <- array(c(1,  1, 1, 1,
                  1 ,  1, 1, 1,
                  1,  1, 1, 1,
                  1,  1, 1),
                dim=c(5,3))

metric = "logEuclidean"

### Different SPD matrix-valued curves (gamma's) ###

ExampleCurve0 <- function(J,p,f.3,f.4){
  n = 2^J
  curve = array(0, dim=c(2,2,n))
  for(k in 1:n){
    curve[,,k] <- matrix(c(f.4((k-1)/n, p), 0, 0, f.4((k-1)/n, p)), nrow=2)
  }
  return(curve)
}

ExampleCurve1 <- function(J,p,f.3,f.4){
  n = 2^J
  curve = array(0, dim=c(2,2,n))
  for(k in 1:n){
    curve[,,k] <- matrix(c(f.4((k-1)/n, p), f.3((k-1)/n, p), f.3((k-1)/n, p), f.4((k-1)/n, p)), nrow=2)
  }
  return(curve)
}

ExampleCurve2 <- function(J,p,f.4,f.5){
```

```
  n = 2^J
  curve = array(0, dim=c(2,2,n))
  for(k in 1:n){
    curve[,,k] <- matrix(c(f.4((k-1)/n, p), f.5((k-1)/n, p), f.5((k-1)/n, p), f.4((k-1)/n, p)), nrow=2)
  }
  return(curve)
}

ExampleCurve4 <- function(J,p,f.4,f.5,f.6){
  n = 2^J
  curve = array(0, dim=c(2,2,n))
  for(k in 1:n){
    curve[,,k] <- matrix(c(f.4((k-1)/n, p), f.5((k-1)/n, p), f.5((k-1)/n, p), f.6((k-1)/n, p)), nrow=2)
  }
  return(curve)
}

ExampleCurve5 <- function(J,p,f.1,f.2,f.3,f.4){
  n = 2^J
  curve = array(0, dim=c(2,2,n))
  for(k in 1:n){
    curve[,,k] <- matrix(c(f.4((k-1)/n, p), f.8((k-1)/n, p), f.8((k-1)/n, p), f.6((k-1)/n, p)), nrow=2)
  }
  return(curve)
}

### Data models ###

## Spectral decomposition model ##
SpecDecompModel <- function(curve){
  n = dim(curve)[3]
  eigenvalues = array(0, dim=c(2,2,n))
  eigenvector = array(0, dim=c(2,2,n))
  newvalues = array(0, dim=c(2,2,n))
  X = array(0, dim=c(2,2,n))
  for (i in 1:n){
    eigenvalues[,,i] <- diag(eigen(curve[,,i], symmetric = TRUE, only.values = FALSE, EISPACK = FALSE)$values) # store the diagonal matrix
        of eigenvalues
    eigenvector[,,i] <- eigen(curve[,,i], symmetric = TRUE, only.values = FALSE, EISPACK = FALSE)$vectors
    newvalues[,,i] <- exp(rnorm(1, 0, 0.1))*eigenvalues[,,i]
    X[,,i] <- eigenvector[,,i] %*% newvalues[,,i] %*% t(eigenvector[,,i])
  }
  return(X)
}

## Signal-plus-noise model ##
GaussianAdditiveModell <- function(curve, sigmap){
  n = dim(curve)[3]
  X = array(0, dim=c(2,2,n))
  for(k in 1:n){
    entries = c(rnorm(1,0,sigma[1]), rnorm(1,0,sigma[3]), rnorm(1,0,sigma[2])) # Gaussian errors
    xi.k = matrix(c(entries[1],entries[2],entries[2],entries[3]), nrow=2)
    X[,,k] <- expm(logm(curve[,,k])+ xi.k)
  }
  return(X)
}

### Simulations ###

## Test PD condition on a matrix ##
TestPositiveDefiniteness <- function(matrix){
  value = 0
  if (matrix[1,1]*matrix[2,2] > matrix[1,2]^2 & (matrix[1,1] + matrix[2,2])>0){
    value = 1
  }
  return(value)
}

## Test PD condition on a curve of matrices ##
TestPositiveDefinitenessCurve <- function(curve){
  value = 1
  n = dim(curve)[3]
  for(k in 1:n){
    value = value*TestPositiveDefiniteness(curve[,,k])
  }
  return(value)
}
```

```
148
149 ## Data based on the singal-plus-noise model ##
150 set.seed(42) # All the graphs found and displayed in the text uses this random generated number
151 curve <- ExampleCurve5(J,p,f.4,f.5,f.6)
152 curve.noise <- GaussianAdditiveModell(curve, sigma[p,])
153 TestPositiveDefinitenessCurve(curve)
154 TestPositiveDefinitenessCurve(curve.noise)
155
156 ## Data based on the spectral decomposition model ##
157 set.seed(42)
158 curve <- ExampleCurve5(J,p,f.4,f.5,f.6)
159 curve.noise <- SpecDecompModel(curve)
160 TestPositiveDefinitenessCurve(curve) # test PD condition
161 TestPositiveDefinitenessCurve(curve.noise) # # test PD condition
162
163 ## Denoising via wavelet (tree-structured) thresholding ##
164
165 Curve.noise.log.tt <- pdSpecEst1D(curve.noise, order = N, metric = metric, periodic=FALSE, alpha=0.06, bias.corr =F)$f
166
167 Curve.noise.rie.tt <- pdSpecEst1D(curve.noise, order = N, metric = "Riemannian", periodic=FALSE, alpha=0.06, bias.corr =F)$f
168
169 Curve.noise.log.st <- SPDEst1D(curve.noise, order = N, metric = metric, periodic=FALSE, alpha=0.65, bias.corr =F)$f
170
171 ## Generating graphs ##
172
173 # Components of the curve #
174 par(mfrow=c(2,2))
175 t = seq(0,1,2^(-J))
176 plot(seq(0,1,2^(-J))[1:2^J], curve[1,1,], col="red", type="s", lwd=2, xlab = "", ylab = "(1,1)" )
177 plot(seq(0,1,2^(-J))[1:2^J], curve[1,2,], col="red", type="s", lwd=2, xlab = "", ylab = "(1,2)" )
178 plot(seq(0,1,2^(-J))[1:2^J], curve[2,1,], col="red", type="s", lwd=2, xlab = "", ylab = "(2,1)" )
179 plot(seq(0,1,2^(-J))[1:2^J], curve[2,2,], col="red", type="s", lwd=2, xlab = "", ylab = "(2,2)" )
180
181 # Components of the noisy curve #
182 par(mfrow=c(2,2))
183 t = seq(0,1,2^(-J))
184 plot(seq(0,1,2^(-J))[1:2^J], curve.noise[1,1,], col="red", type="s", lwd=2, xlab = "", ylab = "(1,1)" )
185 plot(seq(0,1,2^(-J))[1:2^J], curve.noise[1,2,], col="red", type="s", lwd=2, xlab = "", ylab = "(1,2)" )
186 plot(seq(0,1,2^(-J))[1:2^J], curve.noise[2,1,], col="red", type="s", lwd=2, xlab = "", ylab = "(2,1)" )
187 plot(seq(0,1,2^(-J))[1:2^J], curve.noise[2,2,], col="red", type="s", lwd=2, xlab = "", ylab = "(2,2)" )
188
189 ## Estimates with original curve - component (1,1) ##
190
191 par(mfrow=c(1,3))
192 t = seq(0,1,2^(-J))
193 plot(seq(0,1,2^(-J))[1:2^J], Curve.noise.log.tt[1,1,], col="blue", type="s", lwd=2, xlab = "", ylab = "" )
194 lines(f.4(t,p)~t, type="l", lwd=2, col="red")
195
196 plot(seq(0,1,2^(-J))[1:2^J], Curve.noise.rie.tt[1,1,], col="blue", type="s", lwd=2, xlab = "", ylab = "" )
197 lines(f.4(t,p)~t, type="l", lwd=2, col="red")
198
199 plot(seq(0,1,2^(-J))[1:2^J], Curve.noise.log.st[1,1,], col="blue", type="s", lwd=2, xlab = "", ylab = "" )
200 lines(f.4(t,p)~t, type="l", lwd=2, col="red")
201
202 ## Estimates with original curve - component (2,2) ##
203 par(mfrow=c(1,3))
204 t = seq(0,1,2^(-J))
205
206 ### log-eucl old ruke
207 plot(seq(0,1,2^(-J))[1:2^J], Curve.noise.log.tt[2,2,], col="blue", type="s", lwd=2, xlab = "", ylab = "" )
208 lines(f.6(t,p)~t, type="l", lwd=2, col="red")
209
210 plot(seq(0,1,2^(-J))[1:2^J], Curve.noise.rie.tt[2,2,], col="blue", type="s", lwd=2, xlab = "", ylab = "" )
211 lines(f.6(t,p)~t, type="l", lwd=2, col="red")
212
213 plot(seq(0,1,2^(-J))[1:2^J], Curve.noise.log.st[2,2,], col="blue", type="s", lwd=2, xlab = "", ylab = "" )
214 lines(f.6(t,p)~t, type="l", lwd=2, col="red")
215
216
217 ## Estimates with original curve - component (1,2) ##
218 par(mfrow=c(1,3))
219 t = seq(0,1,2^(-J))
220
221 ### log-eucl old ruke
222 plot(seq(0,1,2^(-J))[1:2^J], Curve.noise.log.tt[1,2,], col="blue", type="s", lwd=2, xlab = "", ylab = "" )
223 #plot(seq(0,1,2^(-J))[1:2^J], curve.noise[1,1,], col="grey", type="p", xlab = "", ylab = "f.1" )
224 lines(f.8(t,p)~t, type="l", lwd=2, col="red")
```

```
#lines(seq(0,1,2^(-J))[1:2^J], Curve.noise.f.t[1,1,], type="s", lwd=2, col="red")

plot(seq(0,1,2^(-J))[1:2^J], Curve.noise.rie.tt[1,2,], col="blue", type="s", lwd=2, xlab = "", ylab = "" )
lines(f.8(t,p)^t, type="l", lwd=2, col="red")

plot(seq(0,1,2^(-J))[1:2^J], Curve.noise.log.st[1,2,], col="blue", type="s", lwd=2, xlab = "", ylab = "" )
lines(f.8(t,p)^t, type="l", lwd=2, col="red")


### Accuracy and time computations ###

## Average MSE ##

# Average MSE - log-Euclidean metric - TT rule #
MSELog <- vector("numeric", length = 100)
for (i in 1:100){
set.seed(NULL)
logEst <- array(0, dim=c(2,2,2^J))
logFunc <- array(0, dim=c(2,2,2^J))
errors <- vector("numeric", length = 2^J)
curve.noise <- GaussianAdditiveModell(curve, sigma[p,])
Denoised_LogTT <- pdSpecEst1D(curve.noise, order = N, metric = metric, periodic=FALSE, alpha=0.06, bias.corr =F)$f
for (k in 1:2^J){
  logEst[,,k] <- logm(Re(Denoised_LogTT[,,k]))
  logFunc[,,k] <- logm(Re(curve[,,k]))
  errors[k] <- (norm(logEst[,,k] - logFunc[,,k], type = c("F")))^2
}
MSELog[i] <- (1/2^J)*sum(errors)
}
mean(MSELog)

# Average MSE - affine-onvariant metric - TT rule #

MSERie <- vector("numeric", length = 100)
for (i in 1:100){
  set.seed(NULL)
  logEst <- array(0, dim=c(2,2,2^J))
  logFunc <- array(0, dim=c(2,2,2^J))
  errors <- vector("numeric", length = 2^J)
  curve.noise <- GaussianAdditiveModell(curve, sigma[p,])
  Denoised_RieTT <- pdSpecEst1D(curve.noise, order = N, metric = "Riemannian", periodic=FALSE, alpha=0.06, bias.corr =F)$f
  for (k in 1:2^J){
    logEst[,,k] <- logm(Re(Denoised_RieTT[,,k]))
    logFunc[,,k] <- logm(Re(curve[,,k]))
    errors[k] <- (norm(logEst[,,k] - logFunc[,,k], type = c("F")))^2
  }
  MSERie[i] <- (1/2^J)*sum(errors)
}
mean(MSERie)

# Average MSE - log-Euclidean metric - ST rule #
MSELogST <- vector("numeric", length = 100)
for (i in 1:100){
  set.seed(NULL)
  logEst <- array(0, dim=c(2,2,2^J))
  logFunc <- array(0, dim=c(2,2,2^J))
  errors <- vector("numeric", length = 2^J)
  curve.noise <- GaussianAdditiveModell(curve, sigma[p,])
  Denoised_LogST <- SPDEst1D(curve.noise, order = N, metric = metric, periodic=FALSE, alpha=0.64, bias.corr =F)$f
  for (k in 1:2^J){
    logEst[,,k] <- logm(Re(Denoised_LogST[,,k]))
    logFunc[,,k] <- logm(Re(curve[,,k]))
    errors[k] <- (norm(logEst[,,k] - logFunc[,,k], type = c("F")))^2
  }
  MSELogST[i] <- (1/2^J)*sum(errors)
}
mean(MSELogST)

## Average computation time ##

# log-Euclidean metric - TT rule #
timeLogTT <- vector("numeric", length = 1000)
for (i in 1:1000){
   start_time <- Sys.time()
  denoised <- pdSpecEst1D(curve.noise, order = N, metric = metric, periodic=FALSE, alpha=0.06, bias.corr =F)$f
  end_time <- Sys.time()
  timeLogTT[i] <- end_time - start_time
```

```
}
mean(timeLogTT)

# affine-invariant metric - TT rule #
timeRieTT <- vector("numeric", length = 1000)
for (i in 1:1000){
  start_time <- Sys.time()
  denoised <- pdSpecEst1D(curve.noise, order = N, metric = "Riemannian", periodic=FALSE, alpha=0.06, bias.corr =F)$f
  end_time <- Sys.time()
  timeRieTT[i] <- end_time - start_time
}
mean(timeRieTT)

# log-Euclidean metric - ST rule #
timeLogST <- vector("numeric", length = 1000)
for (i in 1:1000){
  start_time <- Sys.time()
  denoised <- SPDEst1D(curve.noise, order = N, metric = metric, periodic=FALSE, alpha=0.64, bias.corr =F)$f
  end_time <- Sys.time()
  timeLogST[i] <- end_time - start_time
}
mean(timeLogST)
```

Adapted version of the function `pdCART` (Chau 2020) - `spdCART`

```
SPDCart <- function (D, D.white, order, alpha = 1, tree = TRUE, ...)
{
  J <- length(D)
  d <- dim(D[[1]])[1]
  is_2D <- ifelse(length(dim(D[[1]])) == 4, TRUE, FALSE)
  dots <- list(...)
  B <- (if (is.null(dots$B))
    d
    else dots$B)
  periodic <- (if (is.null(dots$periodic) | is_2D)
    FALSE
    else dots$periodic)
  return.D <- (if (is.null(dots$return.D))
    NA
    else dots$return.D)
  w.tree <- (if (is.null(dots$w.tree))
    NULL
    else dots$w.tree)
  if (periodic) {
    L <- (order - 1)/2
    L_b <- ceiling(L/2)
  }
  else {
    L <- L_b <- 0
  }
  lam <- (if (is.null(dots$lam))
    NA
    else dots$lam)
  if (is_2D) {
    D_sum <- lapply(2:J, function(j) apply(D.white[[j]],
                                           c(3, 4), function(A) sum(A)))
    J_sum <- length(D_sum)
    s_e2D <- stats::mad(c(D_sum[[J_sum]]))
    J0_2D <- sum(sapply(1:J, function(j) any(dim(D[[j]]) ==
                                               1)))
    if (J0_2D > 1) {
      n_2D <- which.max(dim(D[[J0_2D]])[c(3, 4)])
      if ((max(order > 9))) {
        stop("The marginal refinement orders in 'order' should all be smaller or equal to 9")
      }
      for (j in 1:length(D_sum)) {
        if (j < J0_2D) {
          n <- dim(D_sum[[j]])[n_2D]
          N <- ifelse(order[n_2D] > n, 2 * floor((n -
                                                    1)/2) + 1, order[n_2D])
          L <- (N - 1)/2
          l <- sapply(1:n, function(k) if ((k - L) <
                                           1)
            2 * k
            else if ((k + L) >= n)
              2 * (N - (n - k))
```

```
            else N + 1)
          s_e1D <- sqrt(sapply(1:n, function(k) 4^(-J) *
                               sum(W_1D[[L + 1]][l[k], ]^2) * sum(trigamma(B -
                                                                          (d - 1:d)))))
          D_trace[[j]] <- D_trace[[j]]/s_e1D
        }
        else {
          D_trace[[j]] <- D_trace[[j]]/s_e2D
        }
      }
    }
    else {
      D_trace <- lapply(1:J_tr, function(j) D_trace[[j]]/s_e2D)
    }
  }
  else {
    D_sum_full <- lapply(2:J, function(j) apply(D.white[[j]],
                                                3, function(A) Re(sum(A))))
    J_tr <- length(D_sum_full)
    sigma_choice <- 0.1
    k <- sqrt(2^(-J))
    s_e <- max(stats::mad(c(D_sum_full[[J_tr]])), 1.03*sqrt(2^(-J))*d*sigma_choice)
    D_sum <- lapply(1:J_tr, function(j) D_sum_full[[j]][L_b +
                                                        1:2^j]/s_e)
  }
  if (is.na(lam)) {
    lam <- alpha * sqrt(2 * log(length(unlist(D_sum))))
  }
  if (tree && is.null(w.tree)) {
    w <- D_sum
    for (j in J_tr:1) {
      if (j == J_tr) {
        w[[j]] <- ifelse(abs(D_sum[[j]]) > lam, TRUE,
                         FALSE)
        R <- pmin(D_sum[[j]]^2, lam^2)
        V <- D_sum[[j]]^2
      }
      else {
        if (is_2D) {
          dims <- dim(D_sum[[j]])
          if (all(dim(D_sum[[min(j + 1, J_tr)]]) >
                  1)) {
            l1 <- t(sapply(1:dims[1], function(i) c(1,
                                                    2) + 2 * (i - 1)))
            l2 <- t(sapply(1:dims[2], function(i) c(1,
                                                    2) + 2 * (i - 1)))
            grid <- expand.grid(1:dims[1], 1:dims[2])
            V <- array(c(mapply(function(i1, i2) sum(V[l1[i1,
            ], l2[i2, ]]), grid$Var1, grid$Var2)),
            dim = dims) + D_sum[[j]]^2
            R <- array(c(mapply(function(i1, i2) sum(R[l1[i1,
            ], l2[i2, ]]), grid$Var1, grid$Var2)),
            dim = dims) + lam^2
          }
          else if (dim(D_sum[[j + 1]])[1] == 1) {
            V <- sapply(1:dims[2], function(i) V[1,
                                                 2 * i - 1] + V[1, 2 * i]) + D_sum[[j]]^2
            R <- sapply(1:dims[2], function(i) R[1,
                                                 2 * i - 1] + R[1, 2 * i]) + lam^2
          }
          else if (dim(D_sum[[j + 1]])[2] == 1) {
            V <- sapply(1:dims[1], function(i) V[2 *
                                                   i - 1, 1] + V[2 * i, 1]) + D_sum[[j]]^2
            R <- sapply(1:dims[1], function(i) R[2 *
                                                   i - 1, 1] + R[2 * i, 1]) + lam^2
          }
        }
        else {
          dims <- length(D_sum[[j]])
          V <- sapply(1:dims, function(i) V[2 * i -
                                              1] + V[2 * i]) + D_sum[[j]]^2
          R <- sapply(1:dims, function(i) R[2 * i -
                                              1] + R[2 * i]) + lam^2
        }
        w[[j]] <- ifelse(R < V, TRUE, FALSE)
        R <- pmin(V, R)
      }
```

```
    }
  }
  else if (!isTRUE(tree) && is.null(w.tree)) {
    w <- lapply(1:J_tr, function(j) abs(D_sum[[j]]) >
                  lam)
  }
  else if (!is.null(w.tree)) {
    w <- w.tree
  }
  D_w <- D
  if (isTRUE(return.D == "D.white")) {
    D.white_w <- D.white
  }
  if (is_2D) {
    for (j in 2:J) {
      if (isTRUE(tree)) {
        if (j > 2) {
          dims <- dim(w[[j - 1]])
          roots <- matrix(rep(matrix(rep(t(w[[j - 2]]),
                                         each = ifelse(dims[2] > 1, 2, 1)), byrow = TRUE,
                                     ncol = dims[2]), each = ifelse(dims[1] >
                                                                      1, 2, 1)), nrow = dims[1])
          w[[j - 1]] <- w[[j - 1]] & roots
        }
      }
      D0 <- array(D_w[[j]], dim = c(d, d, dim(D_w[[j]])[3] *
                                      dim(D_w[[j]])[4]))
      D0[, , !(w[[j - 1]])] <- 0
      D_w[[j]] <- array(D0, dim = c(d, d, dim(D_w[[j]])[3],
                                    dim(D_w[[j]])[4]))
      if (isTRUE(return.D == "D.white")) {
        D0 <- array(D.white[[j]], dim = c(d, d, dim(D.white_w[[j]])[3] *
                                            dim(D.white_w[[j]])[4]))
        D0[, , !(w[[j - 1]])] <- 0
        D.white_w[[j]] <- array(D0, dim = c(d, d, dim(D.white_w[[j]])[3],
                                            dim(D.white_w[[j]])[4]))
      }
    }
  }
  else {
    for (j in 2:J) {
      w[[j - 1]] <- (if (isTRUE(tree))
        w[[j - 1]] & rep((if (j == 2) TRUE else w[[j -
                                                   2]]), each = 2)
        else w[[j - 1]])
      if (periodic & (L_b > 0)) {
        zeros <- !(c(abs(D_sum_full[[j - 1]][1:L_b]) >
                       lam, w[[j - 1]], abs(D_sum_full[[j - 1]][2^(j -
                                                                     1) + L_b + 1:L_b]) > lam))
      }
      else {
        zeros <- !(w[[j - 1]])
      }
      D_w[[j]][, , zeros] <- 0
      if (isTRUE(return.D == "D.white")) {
        D.white_w[[j]][, , zeros] <- 0
      }
    }
  }
  res <- (if (!isTRUE(return.D == "D.white"))
    list(w = w, D_w = D_w)
    else list(w = w, D_w = D_w, D.white_w = D.white_w))
  return(res)
}
```

Adapted version of `pdSpecEst1D` (Chau 2020) - `spdEst1D`

```
SPDEst1D <- function (P, order = 5, metric = "Riemannian", alpha = 1, return_val = "f",
          ...)
{
  dots <- list(...)
  tree <- (if (is.null(dots$tree))
    TRUE
    else dots$tree)
  w.tree <- (if (is.null(dots$w.tree))
    NULL
```

```
    else dots$w.tree)
  periodic <- (if (is.null(dots$periodic))
    TRUE
    else dots$periodic)
  method <- (if (is.null(dots$method))
    "fast"
    else dots$method)
  metric <- match.arg(metric, c("Riemannian", "logEuclidean",
                                "Cholesky", "rootEuclidean", "Euclidean", "Riemannian-Rahman"))
  J <- log2(dim(P)[3])
  d <- dim(P)[1]
  B <- (if (is.null(dots$B))
    d
    else dots$B)
  J.out <- (if (is.null(dots$J.out))
    J
    else dots$J.out)
  jmax <- min((if (is.null(dots$jmax)) J - 2 else dots$jmax),
              J.out - 1)
  bias.corr <- (if (is.null(dots$bias.corr))
    TRUE
    else dots$bias.corr)
  return.D <- (if (is.null(dots$return.D))
    NA
    else dots$return.D)
  P <- (if ((grepl("Riemannian", metric) | metric == "logEuclidean") &
            bias.corr) {
    B * exp(-1/d * sum(digamma(B - (d - 1:d)))) * P
  }
  else P)
  coeff <- WavTransf1D(P, order, jmax = jmax, periodic = periodic,
                       metric = metric, method = method)
  coeff.thresh <- SPDCart(coeff$D, coeff$D.white, alpha = alpha,
                          tree = tree, periodic = periodic, w.tree = w.tree, order = order,
                          B = B, return.D = return.D)
  f <- (if (return_val == "f") {
    InvWavTransf1D(coeff.thresh$D_w, coeff$M0, order = order,
                   jmax = J.out, periodic = periodic, metric = metric,
                   method = method, chol_bias = bias.corr)
  }
  else NULL)
  if (!isTRUE(return.D == "D.white")) {
    res <- list(f = f, D = coeff.thresh$D_w, M0 = coeff$M0,
                tree.weights = coeff.thresh$w, D.raw = coeff$D)
  }
  else {
    res <- list(f = f, D = coeff.thresh$D_w, M0 = coeff$M0,
                tree.weights = coeff.thresh$w, D.raw = coeff$D,
                D.white = coeff.thresh$D.white_w)
  }
  return(res)
}
```

References

Antoniadis, A. (1997). "Wavelets in statistics: a review". In: *Statistical Methods and Applications* 6.2, pp. 97–130.

Arsigny, V. et al. (Jan. 2006). "Geometric Means in a Novel Vector Space Structure on Sysmetric Positive-Definite Matrices". In: *SIAM J. Matrix Analysis Applications* 29, pp. 328–347. DOI: 10.1137/050637996.

Bhatia, R. (2006). "Positive Definite Matrices". In: *Princeton University Press* New Jersey.

Bihan, D. Le (1991). "Diffusion MNR imaging". In: *Magnetic Resonance Quarterly* 7, pp. 1–30.

Boothby, W. M. (1986). *An Introduction to Differentiable Manifolds and Riemannian Geometry*. Academic Press, New York.

Boumal, N. and P-A. Absil (2011). "Discrete regression methods on the cone of positive-definite matrices". In: *IEEE ICASSP*, pp. 4232–4235.

Carmo, M.P. do (1992). *Riemannian Geometry*. Birkhäuser, Boston.

Chau, J. (2018). "Advances in Spectral Analysis for Multivariate, Nonstationary and Replicated Time Series". In: *Université catholique de Louvain - Institute of Statistics, Biostatistics and Actuarial Sciences* PhD Thesis.

— (2020). "pdSpecEst: An Analysis Toolbox for Hermitian Positive Definite Matrices". Version package version 1.2.4. In: *The CRAN*. DOI: https://CRAN.R-project.org/package=pdSpecEst.

Chau, J. and R. von Sachs (2016). "Functional mixed effects wavelet estimation for spectra of replicated time series". In: *Electronic Journal of Statistics Journal of Statistics* 10(2), pp. 2461–2510.

— (2021). "Intrinsic wavelet regression for curves of Hermitian positive definite matrices". In: *Journal of the American Statistical Association* 116(534), pp. 819–832.

Donoho, D.L. (1997). "Cart and best-ortho-basis: a connection". In: *The Annals of Statistics* 25.5, pp. 1870–1911.

— (1993). *Smooth wavelet decompositions with blocky coefficient kernels*. Recent Advances in Wavelet Analysis, pages 259–308. Academic Press, New York. ISBN: 9781681730134.

Dryden, I.L., A. Koloydenko, and D. Zhou (2009). "Non-Euclidean statistics for covariance matrices, with applications to diffusion tensor imaging". In: *The Annals of Applied Statistics* 3.3, pp. 1102–1123.

Freyermuth, J.M. (2011). "Tree-structured Wavelets in Nonparametric Function Estimation". In: *PhD thesis, Université catholique de Louvain*.

Hall, B. (2003). *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*. Graduate Texts in Mathematics 222. Springer. ISBN: 97803872155499.

Hinkle, J., P.T. Fletcher, and S. Joshi (2014). "Intrinsic polynomials for regression on Riemannian manifolds". In: *Journal of Mathematical Imaging and Vision* 50.(1-2), pp. 32–52.

Huang, Z. (2015). "Log-Euclidean Metric Learning on Symmetric Positive Definite Manifold with Application to Image Set Classification". In: *JMLR: WCP* 37.

Jansen, M. (2001). *Noise Reduction by Wavelet Thresholding*. Springer-Verlag, New York.

Jansen, M. and P. Oonincx (2005). *Second Generation Wavelets and Applications*. Springer-Verlag, London.

Jung, S. and A. Schwartzman (June 2014). "Scaling-Rotation Distance and Interpolation of Symmetric Positive-Definite Matrices". In: *SIAM Journal on Matrix Analysis and Applications* 36. DOI: 10.1137/140967040.

Klees, R. and R. Haagmans (2000). *Wavelets in the Geosciences*. Volume 90. Springer Science Business Media. ISBN: 97803872155499.

Lee, J. M. (2013). *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics, 128. Springer, New York.

Leemans, A. and D. K. Jones (2009). "The B-matrix must be rotated when correcting for subject motion in DTI data". In: *Magn. Reson. Med.* 61(6), pp. 1336–1349. DOI: 10.1002/mrm.21890.

Lin, Z., H. Muller, and B. Park (2020). "Additive Models for Symmetric Positive-Definite Matrices, Riemannian Manifolds and Lie groups". In: *arXiv: Methodology*.

Moakher, Maher and Philipp G. Batchelor (2006). "Symmetric Positive-Definite Matrices: From Geometry to Applications and Visualization". In: *Visualization and Processing of Tensor Fields*. Ed. by Joachim Weickert and Hans Hagen. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 285–298. ISBN: 978-3-540-31272-7. DOI: 10.1007/3-540-31272-2_17. URL: https://doi.org/10.1007/3-540-31272-2_17.

Mohammadi, B., H. Borouchaki, and P. L. George (1997). "Delaunay mesh generation governed by metric specifications. II. Applications". In: *Finite Elem. Anal. Des. as*, pp. 85–109.

Nason, G. (2008). *Wavelet Methods in Statistics with R*. Springer, New York.

Nason, G.P. (1996). "Wavelet shrinkage using cross-validation". In: *Journal of the Royal Statistical Society* Series B.58, pp. 463–479.

Pasternak, O., N. Sochen, and P.J. Basser (2010). "The effect of metric selection on the analysis of diffusion tensor MRI data". In: *NeuroImage* 49.3, pp. 2190–2204.

Pennec, X. (2006). "Intrinsic statistics on Riemannian manifolds: Basic tools for geometric measurements". In: *Journal of Mathematical Imaging and Vision* 25.1, pp. 127–154.

Pennec, X., P. Fillard, and N. Ayache (2006). "A Riemannian framework for tensor computing". In: *International Journal of Computer Vision* 66.1, pp. 41–66.

Quang, M. Hà and M. Vittorio (2018). *Covariances in Computer Vision and Machine Learning*. Synthesis Lectures on Computer Vision. Morgan Claypool publishers. ISBN: 9781681730134.

Rahman, I.U. et al. (2005). "Multiscale representations for manifold-valued data". In: *Multiscale Modeling and Simulation* 4.4, pp. 1201–1232.

Rossmanith, G. (2013). *Non-linear Data Analysis on the Sphere: The Quest for Anomalies in the Cosmic Microwave Background*. Springer. ISBN: 978-3319003085.

Said, S. et al. (2017). "Riemannian Gaussian distributions on the space of symmetric positive definite matrices". In: *IEEE Transactions on Information Theory* 63.4, pp. 2153–2170.

Schwartzman, A. (July 2014). "Lognormal Distributions and Geometric Averages of Symmetric Positive Definite Matrices". In: *International Statistical Review* 84. DOI: 10.1111/insr.12113.

Yuan, Y. et al. (2012). "Local polynomial regression for symmetric positive definite matrices". In: *Journal of the Royal Statistical Society: Series B* 74.4, pp. 697–719.

Zhu, H. et al. (2009). "Intrinsic regression models for positive-definite matrices with applications to diffusion tensor imaging". In: *Journal of the American Statistical Association* 104.487, pp. 1203–1212.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
Faculté des sciences
Place des sciences, 2 bte L6.06.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/sc**