POLITECNICO DI TORINO

Course of Mathematical Engineering

Master's degree thesis

Bayesian size-and-shape regression modeling

Supervisors: Prof. Enrico Bibbona, Prof. Gianluca Mastrantonio
Candidate: Gabriele Sega

Academic year 2023-2024

A special thanks to everyone that supported me during this years.
First of all, thanks to my parents Damiana and Giancarlo, that always believed in my choices.
Thanks to my uncle Giorgio, that guided me through difficult situations.
Thanks to my friends, that never let me down when difficuties arised.
Thanks to all my students, that during these years inspired and motivated me to never give up.
Thanks to my karate Sensei, Emanuel Giordano, and to all my training comrades for lightening my burdens during our training sessions.
Thanks to my teachers and to my colleagues.
And a special thanks to my beloved, Martina: a reliable companion that shared with me both joy and struggles, during this incredible journey.

Abstract

Statistical shape analysis is a well-known branch of statistics that aims at inferring on objects' shape. The aim of this work is to implement already known methods, with some slight modifications when needed, and use them to make some simulations on synthetic and real datasets. This branch of statistics involves many other topics in mathematics, such as linear algebra, Markov Chains, MCMC algorithms and calculus.

At first, some examples are presented: many practical applications of statistical shape analysis come from the biological framework and are useful to justify the need for such flexible methods that will be presented during the work. Secondly, the problem of defining a correct mathematical framework for size-and-shape will be assessed, by reporting and reviewing the available literature, with particular care to Mardia's work. Having set such a framework, it will be discussed how to derive a stochastic model for size and shape, that will be then used to set up a Bayesian framework to perform statistical analysis of size-and-shape configurations.

The Bayesian inference is done by means of MCMC algorithms implemented in Julia, using a latent variable approach to correctly isolate the size-and-shape information from the rotational one, pointing out and solving any identification issue of the parameters that might arise. This kind of approach has been already explored in literature and allows for a flexible analysis of the problem. More specifically, synthetic datasets will be generated in various configurations to assess the performances of the model in both the two-dimensional and three-dimensional case. The latter case will be treated with particular care, as some slight modifications are proposed with respect to standard work: we model the rotation information by means of Euler angles and explicitly derive the angles' full conditionals, rather than just using a metropolis step, already proposed in previous studies. This is achieved by further developing some known relations, such as the distribution of the rotations being Matrix Fisher, obtaining two Von-Mises angles and a third angle that can be sampled by using specific accept-reject methods.

The performances of the model are then assessed by means of specific metrics, such as the Riemannian distance, allowing us to make comparisons between estimates and real data. The work shows how the Bayesian approach can be implemented with little assumptions, leading to remarkable results, with affordable computational efforts, thanks to a proper code implementation.

Contents

Chapter 1

Statistical shape analysis

Everything surrounding us has a shape. Since the early years of life, we are able to recognize and distinguish objects that have different shapes.
Studying this concept has both theoretical and practical implications: the former, is to define such an abstract concept in mathematical terms, the latter is to use the knowledge of this concept to achieve useful results in different branches.
Before starting with the practical examples, it is useful to introduce some mathematical definitions.

1.1 Previous studies

The study of the statistical shape analysis is rooted in early 1970s. The first work, as cited in [10], is the one of Mardia (1972) [9] that aimed at study statistics of directional data.
The work was then improved by Mardia and Jupp and finally ended up in a recent book named Directional statistics, published in 2000 [7]. Finally, a new book called "Statistical shape analysis" was published in 2016 by Mardia and Dryden [10]. The latter will be one of the main reference point for this thesis work.
These studies all comes from the necessity of dealing with data that retains some directional information: vectors, planes, shapes and many others. In this work, we will revise many of these topics and relate them with other techniques (such as the Bayesian MCMC framework) that allows for a more straightforward implementation of the study cases.
Recently another article by Dryden et al. has been published, in which the authors explore the possibility of analyzing human movement data based on the framework of the directional statistics/ statistical shape analysis. Today,

these models have been used to study many interesting cases using linear regression into the shape space, by including both continuous and factorial covariates when needed.

1.2 The definition of shape

According to Kendall (1977) [8] "shape is everything that remains once location, scale and rotation effects are removed". This means that shape is independent from the reference system and from the scale, which sounds familiar with our everyday experience.
This definition is reasonably in accord with our practical experience: a triangle is such independently on its size, location and rotation.
There are many ways to remove those information and they will be discussed later.
In a similar way, one could point out that the scale size and shape information is everything that remains when rotation and translation information are removed. This is an important key point: these methods allow us to choose at which level of detail we make the inference. If the shape information alone is enough, one could remove the scale. Otherwise if for the use case considered the shape information alone isn't enough, one could also retain scale.
We could repeat a similar process to choose which components to retain: one might be interested in retaining reflection information, while others may not. this will be particularly useful in a Bayesian framework: by choosing which components to retain, we will let the model freely explore the posterior distribution in such a way that those components are retained.
Of course, to define a shape, we need some fixed reference points. This can be done by introducing **landmarks**.

1.3 Landmarks

An important point when dealing with computational approaches to shape is the choice of points that describe the object of the inference. The main idea is that objects with the same shape should share some "reference" points.
As an example, we might think of a human face: the eyes, mouth and nose location can be used as reference points to asses the shape of the face (of course, in a really simplified framework). This intuitive description of shape's points allow us to introduce the definition of what is called a "landmark"

Definition 1.3.1. A landmark is a point of correspondence that matches

between and within populations

Of course there might be many different types of landmarks, depending on the application.
As a final remark, we might also introduce labeled landmarks. We can distinguish **scientific landmark** (also known as **anatomical landmark** when dealing with biological applications), i.e. reference points assigned by an expert that corresponds between objects in some scientifically meaningful way, **mathematical landmarks**, that are located by exploiting the geometrical properties of the object and finally **pseudo-landmarks**. These latter points are located around the previously mentioned landmarks and are usually obtained by construction.
An example of pseudo landmark could be derived by considering the mid points on the outline of an object, between either mathematical or scientific landmarks.
A further improvement could be the assignment of a label to each landmark

Definition 1.3.2. A label is a name or a number associated with a landmark, and identifies which pairs of landmarks correspond when comparing two objects. Such landmarks are called labeled landmarks.

In this way, the correspondence between points in similar objects is exploited in a much clearer way.

1.4 Examples

There are several well known examples,as reported in [10], that might explain efficiently why a statistical shape analysis might be needed.

1.4.1 Mouse vertebrae

A first example of shape analysis is pretty straightforward: suppose that we want to asses the effect of weight in reshaping the vertebrae of a mouse. After dividing mice into 3 groups (Large, small and control), samples of the vertebrae can be collected and then studied.
Free data is available in the R package *shapes* and consists of a total of 76 samples, each containing 60 landmarks points and belonging to one specific group. In this pretty simple framework, it is then possible to compare different shapes configuration according to some statistical methods that will be discussed later.

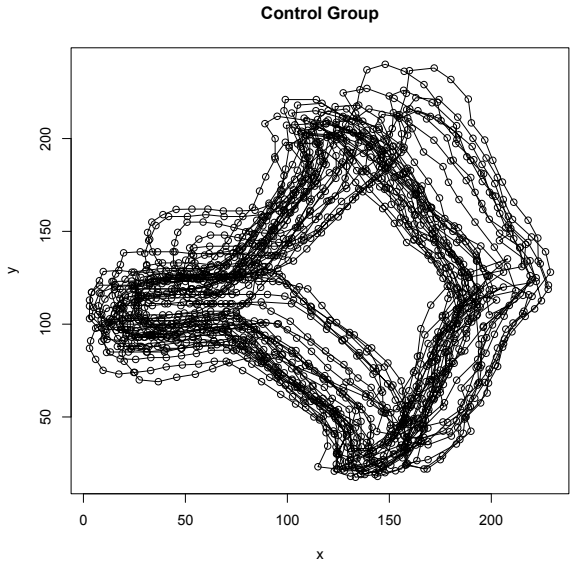
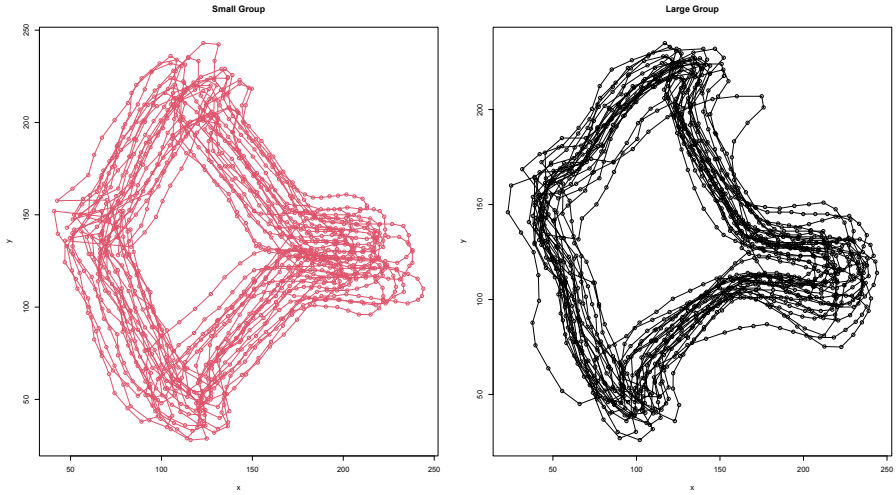


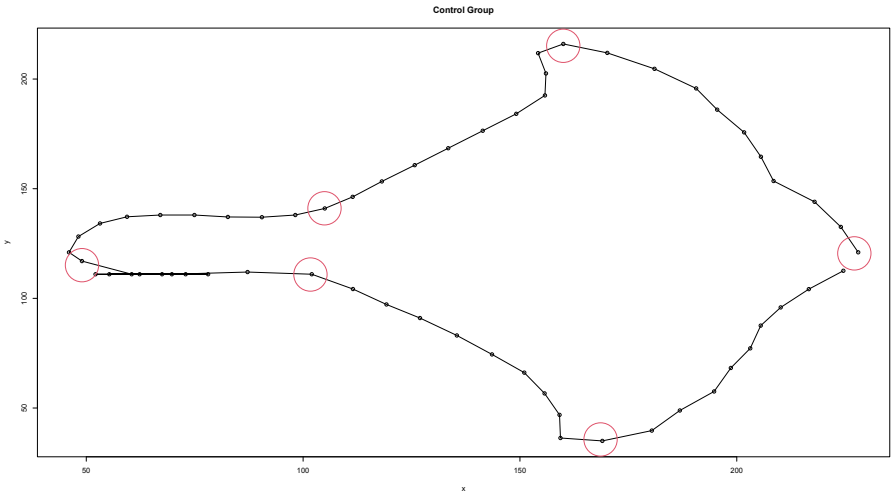
Figure 1.1: Vertebrae samples for control group

As the figure above shows, the samples are composed of 6 landmarks and 54 pseudo-landmarks.
In the following image we can also examine the configurations for the other 2 groups



The first observation that comes at a glance is indeed the reflection orientation: the vertebrae sure have similar shapes, but the control group is reflected with respect to the other 2 configurations. When dealing with simi-

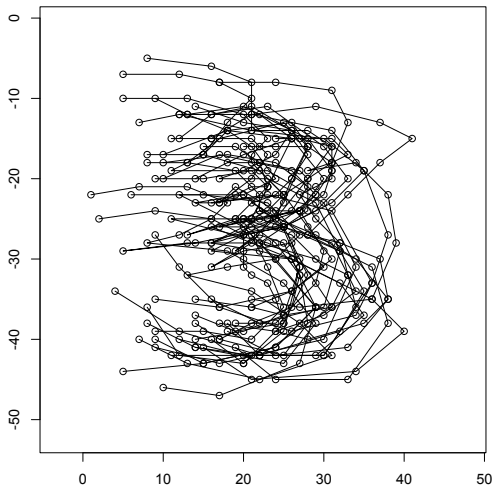
lar datasets, it is always important to fully understand if reflection invariance is needed or not.
The datasets also give a nice explanation of what the difference between landmarks and pseudo landmark is.



The red-circled points are the landmarks, whereas the other are pseudo landmarks that allow to better outline the object's profile.
The use of pseudo-landmarks is indeed recommended to better understand the dataset, however it increases the dimension of the problem and, as a consequence, heavily impacts the computational performances of the inference process.

1.4.2 Image recognition

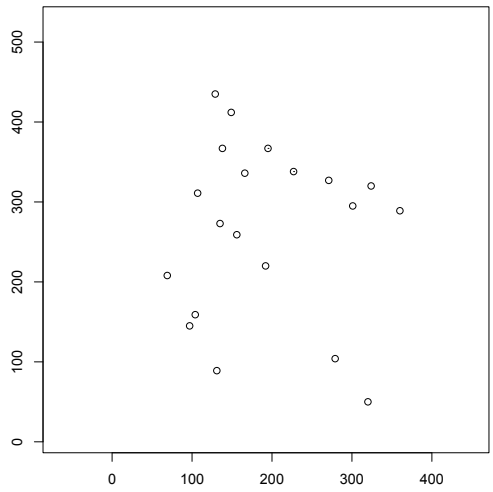
Image recognition is a well known task in the machine learning/ deep learning area of study. Right now, many advanced techniques are available to recognize images of many kinds: starting from a simple multilayer perceptron, up to more complex models based on Convolutional Neural Networks (CNNs). The main issue relative to those mdoels is indeed their accountability: interpreting the output of a Neural network is everything but simple.
An alternative to these approach, when the dimension of the problem is not too big, is using classical statistical methods. Statistical shape analysis could be easily included in such methods and might allow for a lightweight inference. The clear disadvantage s that we need landmarks on images, which are generally reported by hand (that is essentially a manual feature extraction). Here follows some examples take from the dataset *digit3* from the same library *shapes* already mentioned



1.4.3 Electrophoretic gels

As mentioned in [10], another plausible application of shape analysis could be the comparison of electrophoretic gels.
Those gels are the result of a technique for the identification of proteins, and there might be many reasons for which an electrophoretic gel needs to be compared to other by statistical means.

For instance, one might want to identify a particular strain of a certain parasite/ bacteria, being able to sot which points are common and which are not: some points will be present in the gel of every parasite, while some variant points will be different from strain to strain. Even if less visible at a glance, here we report a plot taken from the dataset *gel*



As can be seen from this picture, shape is not always well-interpretable. Taken out from their context, these points might mean nothing. This is a further reason to seek for a mathematical model that is able to define and compare shapes by statistical means: even when shapes are not visible at a glance, we are still able to infere something about their structure. This is the power of mathematical models: the ability of generalizing concrete concepts by abstract means.

1.4.4 Other examples

There are plenty of other examples that might be discussed, and can be found in [10], in many different area of application. Some examples are micro fossils average shape estimation, cortical surface shape comparison between schizophrenic and healthy patients, human movement data recognition.
All this topics might seem distant one from each other, but they all rely to the same paradigm: the information is contained in the shape and size of the object that we are considering.

It comes natural from the previous examples that performing a statistical shape analysis is much more flexible than a simple geometric analysis of the problem. If we think at the first example, the shape of mice vertebrae will indeed be affected also by slightly individual differences. As a consequence, two vertebrae will look different even if the weight is the same. How can we than distinguish the random effect within the population from the random effects between populations? The answer is pretty straightforward: perform-

ing a statistical shape analysis of the problem will not only identify the effect of weight, but it will also quantify how the covariates will influence the mean configuration, in the exact usual way of a typical regression.
In the following, some pictures of other datasets from the *shapes* library are reported

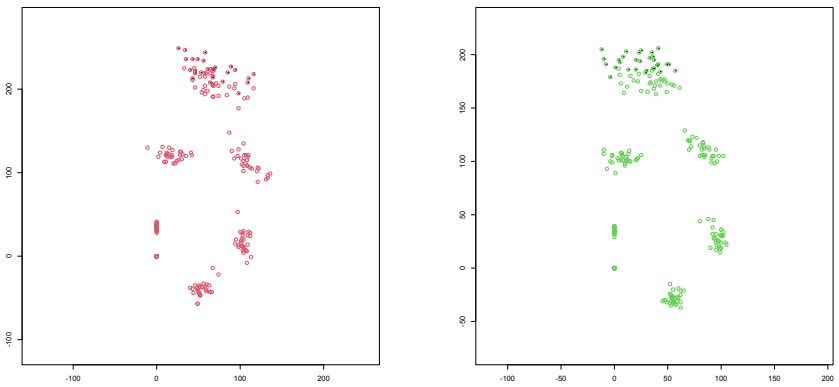


Figure 1.2: Datasets gorm and gorf: male and female gorillas' skulls (left and right, respectively). Each configuration consists of 8 landmarks, having 29 males individual and 30 females individuals

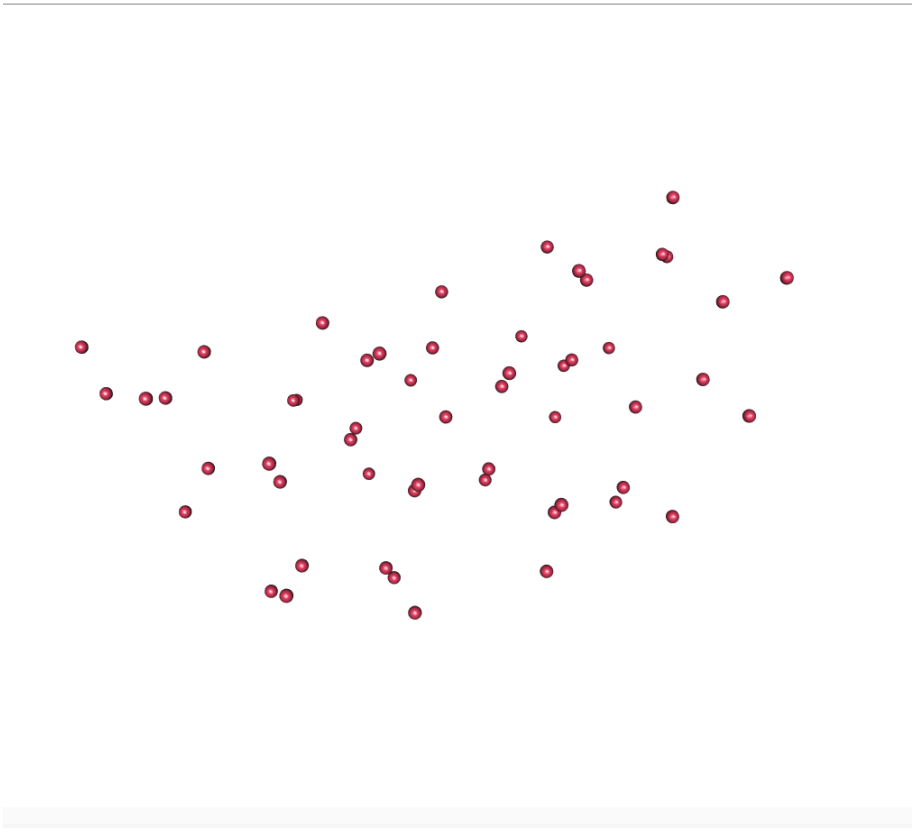


Figure 1.3: example of a steroid molecule from the dataset *steroids*. Each molecule consists of 61 landmarks, representing the atoms of the molecule. A total of 31 molecules is contained in the dataest

.

Chapter 2

Preliminaries

2.1 Distributions background

In the following, we will use a Bayesian approach to perform shape and size inference. Of course this would have never been possible without a solid probability framework for the shape and size space. We will briefly discuss the most important derivations that will later be useful in a MCMC setting, recalling the used distributions and deriving their density when related to the work.

2.1.1 Matrix normal distribution

Let $\mathbf{X} \in \mathbb{R}^{K,p}$ be a random matrix, i.e. a matrix whose entries are random variables. We say that $\mathbf{X}$ follows a matrix normal distribution if its vectorization, which consists of a vector obtained by stacking all the columns of $\mathbf{X}$, follows a multivariate normal distribution. More precisely

Definition 2.1.1.

$$\boldsymbol{X} \sim N_{K,p}(\boldsymbol{\mu}, \boldsymbol{I}_p \otimes \boldsymbol{\Sigma}) \Leftrightarrow \mathrm{vec}(\mathbf{X}) \sim N_{Kp}(\mathrm{vec}(\boldsymbol{\mu}), \boldsymbol{I}_p \otimes \boldsymbol{\Sigma})$$

The pdf for a normal random matrix is given by

$$f(\boldsymbol{X}; \boldsymbol{\mu}, \boldsymbol{I}_p \otimes \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^{kp} \det(\boldsymbol{\Sigma})^p}} \exp\left(-\frac{1}{2}\left[\mathrm{tr}(\boldsymbol{X} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{X} - \boldsymbol{\mu})^T \right] \right)$$

From a practical point of view, we will prefer using the vectorized form rather than the matrix normal itself. The reason behind the choice of including such a distribution in the background is pretty simple to understand: studying the shape of an object means studying a configuration matix whose entries

might be seen as random variables.
Nevertheless this distribution is used by Dryden et al. to model a noticeable use of case scenario [4]

2.1.2 Matrix Fisher distribution

We will now focus on the Matrix Fisher distribution: it is a probability distribution with support in $O(p)$, the orthogonal group of matrices.
Matrix Fisher was first introduced in [5], and then reported also in [1] [7], sometimes referred to as Matrix Langevin. The distribution is defined by the density

$$f(\mathbf{X}) = a(\mathbf{F}) \exp\left\{ \operatorname{tr} FX^T \right\}, \mathbf{X} \in O(p),\ \mathbf{F} \in \mathbb{R}^{n,p}$$

where $\mathbf{F}$ is a parametric matrix and a is a normalization constant. This distribution was firstly introduced by Downs [5] and is one of the first examples of non-uniform distributions on Stiefel-Manifolds (which are not presented here as they are far beyond the purpose of this work).
This family of distributions will be particularly used as they model the random behaviour of some matrices that are involved when dealing with size-and-shape configurations. This can be better seen in the following section, where the SVD for a Matrix Normal variable is derived.

2.1.3 SVD decomposition of a Matrix Normal r.v.

An important key point is how to derive the probability distribution of each component of the SVD decomposition [4]. This will be largely used during the Bayesian inference in the next chapters, as SVD will be one of the most effective ways for dealing with size-sand-shape.
The following theorem holds

Theorem 1. *Let's consider a configuration* $\boldsymbol{X} \in \mathbb{R}^{K,p}$ *and its SVD decomposition* $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{R}^T$. *Assume that* $\boldsymbol{R} \in SO(p)$ *The joint pdf of* $\boldsymbol{U}$ *and* $\boldsymbol{\Delta}$ *is given by*

$$f(\boldsymbol{U}, \boldsymbol{\Delta}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{D(\boldsymbol{\Delta})C(\boldsymbol{A})}{(2\pi)^{\frac{Kp}{2}} |\boldsymbol{\Sigma}|^{\frac{p}{2}}} \exp\left\{ -\frac{1}{2} \operatorname{tr}\left(\boldsymbol{\Delta}\boldsymbol{U}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{U}\boldsymbol{\Delta} + \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} \right) \right\}$$

and the conditional pdf of $\boldsymbol{R}$ *is*

$$f(\boldsymbol{R}|\boldsymbol{A}) = C(\boldsymbol{A})^{-1} \exp\left\{ \operatorname{tr} \boldsymbol{R}\boldsymbol{A}^T \right\}$$

with

$$\boldsymbol{A} = \boldsymbol{\mu}^T \boldsymbol{\Sigma}^{-1} U \boldsymbol{\Delta}$$

$$C(\boldsymbol{A}) = \int_{SO(p)} \exp\{\mathrm{tr}(\boldsymbol{R}\boldsymbol{A}^T)\} d\boldsymbol{R}$$

$$D(\Delta) = 2^{1-p} \det(\Delta) \prod_{i<j}^{p} (\delta_i^2 - \delta_j^2)$$

Proof. The density $f(\boldsymbol{U}, \boldsymbol{\Delta} : \boldsymbol{\mu}, \boldsymbol{\Sigma})$ can be derived by marginalization:

$$f(\boldsymbol{U}, \boldsymbol{\Delta}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = D(\boldsymbol{\Delta}) \int_{SO(p)} f(\boldsymbol{U}, \boldsymbol{\Delta}, \boldsymbol{R}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{R} =$$

$$= D(\boldsymbol{\Delta}) \int_{\boldsymbol{R} \in SO(p)} \frac{1}{\sqrt{(2\pi)^{kp} det(\boldsymbol{\Sigma})^p}} \times$$

$$\times \exp\{-\frac{1}{2} \mathrm{tr}[(\boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{R}^T)^T - \boldsymbol{\mu}\boldsymbol{\Sigma}^{-1}(\boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{R}^T - \boldsymbol{\mu})]\} d\boldsymbol{R} =$$

$$= \frac{D(\boldsymbol{\Delta})}{\sqrt{(2\pi)^{kp} det(\boldsymbol{\Sigma})^p}} \times$$

$$\times \int_{\boldsymbol{R} \in SO(p)} \exp\{-\frac{1}{2} \mathrm{tr}[(\boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{R}^T - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{R}^T - \boldsymbol{\mu})]\} d\boldsymbol{R} =$$

$$= \frac{D(\boldsymbol{\Delta})}{\sqrt{(2\pi)^{kp} det(\boldsymbol{\Sigma})^p}} \times$$

$$\times \int_{\boldsymbol{R} \in SO(p)} \exp\{-\frac{1}{2} \mathrm{tr}[(\boldsymbol{R}\boldsymbol{\Delta}\boldsymbol{U}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{R}^T + \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}]\} \times$$

$$\times \exp\{\mathrm{tr}(\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{U}\Delta\boldsymbol{R}^T)\} d\boldsymbol{R}$$

Now, by using the trace operator properties, we can write

$$\mathrm{tr}(\boldsymbol{R}\boldsymbol{\Delta}\boldsymbol{U}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{R}^T) = \mathrm{tr}(\boldsymbol{R}^T\boldsymbol{R}\boldsymbol{\Delta}\boldsymbol{U}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{U}\boldsymbol{\Delta}) = \boldsymbol{\Delta}\boldsymbol{U}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{U}\boldsymbol{\Delta}$$

And by substituting this result into the previous relation we obtain

$$\frac{D(\boldsymbol{\Delta})}{\sqrt{(2\pi)^{kp} det(\boldsymbol{\Sigma})^p}} \exp\{-\frac{1}{2} tr[(\boldsymbol{\Delta}\boldsymbol{U}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{U}\boldsymbol{\Delta} + \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\} \times$$

$$\int_{\boldsymbol{R} \in SO(p)} \exp\{tr(\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{R}^T)\} d\boldsymbol{R} =$$

$$= \frac{D(\boldsymbol{\Delta})}{\sqrt{(2\pi)^{kp} det(\boldsymbol{\Sigma})^p}} \exp\{-\frac{1}{2} \mathrm{tr}[(\boldsymbol{\Delta}\boldsymbol{U}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{U}\boldsymbol{\Delta} + \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu})]\} \times$$

$$\times \int_{\boldsymbol{R}\in SO(p)} \exp\{\mathrm{tr}(\boldsymbol{R}\boldsymbol{A}^T)\}d\boldsymbol{R} =$$

$$\frac{D(\boldsymbol{\Delta})C(\boldsymbol{A})}{\sqrt{(2\pi)^{kp}det(\boldsymbol{\Sigma})^p}} \exp\{-\frac{1}{2}tr[(\boldsymbol{\Delta}\boldsymbol{U}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{U}\boldsymbol{\Delta} + \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu})]\}$$

□

As a consequence, it is also easy to obtain the conditional distribution of R. Formally, one can write

$$f(\boldsymbol{R}|\boldsymbol{U},\boldsymbol{\Delta}) = \frac{f(\boldsymbol{U},\boldsymbol{\Delta},\boldsymbol{R};\boldsymbol{\mu},\boldsymbol{\Sigma})}{f(\boldsymbol{U},\boldsymbol{\Delta};\boldsymbol{\mu},\boldsymbol{\Sigma})} = \frac{1}{C(\boldsymbol{A})}\exp\{\mathrm{tr}(\boldsymbol{R}\boldsymbol{A}^T)\}$$

which means that $\boldsymbol{R}$, conditioned to $\boldsymbol{A}$, is Matrix Fisher distributed. Further details concerning the normalization constants computation can be found in [4].

2.2 MCMC algorithms

2.2.1 A brief introduction to MCMC algorithms

The key point of the work is the usage of so called MCMC algorithms. The acronym MCMC stands for Markov Chain Monte Carlo, as this class of algorithms is based on both Monte Carlo methods and Markov chains.
The general idea behind MCMC algorithms is to use Monte Carlo methods to simulate a specifically designed Markov chain, in such a way that its limit (stationary) distribution is coincident with the distribution we want to sample from.
In the following, we will briefly recall the most important concepts that allow these methods to work.

2.3 Markov Chains

To fully understand how a MCMC sampler works, we first recall the some useful definitions.

Definition 2.3.1. Discrete time Markov chain
A discrete time Markov chain is a stochastic process $\{X_1,\dots,X_T\}$ with the **Markov Property**, i.e.

$$\mathbb{P}\{X_{t+1} = x|X_t = x_t, X_{t-1} = x_{t-1},\dots,X_1 = x_1\} = \mathbb{P}\{X_{t+1} = x|X_t = x_t\}$$

One interesting fact is that the dynamics of the system can be described by algebraic means by defining a transition probability matrix $\boldsymbol{P}$, i.e. a matrix whose entries satisfy

$$p_{ij} = \mathbb{P}\{X_{t+1} = j | X_t = i\}$$

In this way, given a probability distribution π_t over the states of the chain at time time t, the probability distribution over the states at time $t+1$ can be obtained by post multiplying by P

$$\pi_t P = \pi_{t-1}$$

It must be noticed that those probabilities depend, in general, also from time. This won't be the case for the applications we will consider, and therefore we will only deal with stationary transition probabilities.
One natural question that one might arise is if there exists a distribution such that the system becomes stationary. This is equal to ask if it exist a so called stationary distribution for the system. The latter can be defined in the following way

Definition 2.3.2. Stationary distribution
For a given (Discrete time) Markov chain, a stationary distribution π is a probability distribution over the states of the chain that satisfies

$$\pi P = \pi \tag{2.1}$$

This shouldn't be confused with the *limit* distribution

Definition 2.3.3. For a given Markov chain, the limit distribution is obtained by solving

$$\lim_{t \to \infty} \pi_t \tag{2.2}$$

The two distributions might resemble the same thing, but in general they're not. It is clear that, if it exists, a limit distribution is also stationary, but the converse is not necessarily true.
As a matter of fact, this is true only under specific assumptions. In the following we will consider only the case where the limit distribution exists.
Stationary distributions are of vital importance in many scenarios: from Markov decision processes, to queue theory. For these reasons, they are well studied and some noticeable properties are known. Above all, one might recall the detailed balance.

Definition 2.3.4. A distribution is detailed balance if for any couple of reachable states of the chain the incoming probability fluxes are equal to the outcoming probability fluxes. In mathematical terms it reads

$$\forall i, j \in \mathcal{S} \quad \mathbb{P}\{i \to j\}\pi(i) = \mathbb{P}\{j \to i\}\pi(j) \tag{2.3}$$

Or in a matrix notation

$$P_{ji}\pi_j = P_{ij}\pi_i$$

As a consequence, one might prove the following

Theorem 2. *Any detailed balance distribution is stationary.*

Proof. By definition of stationary distribution

$$\pi = \pi P \tag{2.4}$$

Then, by the detailed balance, we have that

$$P_{ij}\pi_i = P_{ji}\pi_j$$

Summing both sides on i

$$\sum_i P_{ij}\pi_i = \sum_i P_{ji}\pi_j = \pi_j$$

where the last equality comes from the fact that P is row-stochastic.
Finally we obtain

$$(\pi P)_j = \pi_j$$

which concludes the proof □

2.4 MCMC algorithms

The theory behind Markov Chains can be used to simulate from a specific distribution.
Suppose that we want to simulate a sample from a distribution with density $f(x)$. Suppose also that the last sample obtained, let's say x_b, conditioned to all the simulated samples, depends only on its previous sample. In formulae

$$f(x_b|x_{b-1}, x_{b-2}, \ldots, x_0) = f(x_b|x_{b-1})$$

Which means asking that the samples' chain has the Markov property.
Let now $T(x_b|x_{b-1})$ be the transition probability from x_{b-1} to x_b. We aim at

finding T in such a way that the stationary distribution of the Markov Chain coincides with f.
By using the previous results, we ask for f to follow the detailed balance. In formulae

$$T(x_{b-1}|x_b)f(x_b) = T(x_b|x_{b-1})f(x_{b-1})$$

Different choices of T will result in different algorithms. In the following we will discuss the two main algorithms in MCMC simulation: the Gibbs algorithm and the Metropolis-Hastings algorithm.

2.5 Gibbs sampler

The first well-known algorithm is indeed the Gibbs algorithm.
Suppose to want to simulate from a density $f(\mathbf{x}|\theta)$, where $\mathbf{x} = (x_1, \ldots, x_p)$ and $\theta = (\theta_1, \ldots, \theta_m)$ is a vector of parameters.
The idea is to sample at each component $x_i$from its **full conditional**, that is the distribution of x_i conditioned to all others components. In a MCMC setting this means:

Algorithm 1 Gibbs sampler

1: $b = 0$
2: **while** b<B **do**
3: $\quad b \leftarrow b + 1$
4: $\quad$ **for** $i < n$ **do**
5: $\quad\quad \boldsymbol{x_i^b} \sim \boldsymbol{X_i^b}|\boldsymbol{x_1^b}, \boldsymbol{x_2^b}, \boldsymbol{x_{i-1}^b}, \ldots, \boldsymbol{x_{i+1}^{b-1}}, \ldots, \boldsymbol{x_n^{b-1}}$
6: $\quad$ **end for**
7: $\quad \boldsymbol{x}^b \leftarrow (\boldsymbol{x_1^b}, \ldots, \boldsymbol{x_n^b})$
8: **end while**
9: **return** $\boldsymbol{X} = (\boldsymbol{x^1}, \ldots, \boldsymbol{x^B})$

It is worth noting that for a given component i at iteration b we use all the available information, i.e. we use the just sampled values for the components $1, \ldots, i-1$.
Finally, one can obtain a sample from the joint distribution by putting all the just computed samples together. In other words let $x_1^b, \ldots, x_n^b$ be the samples obtained by Gibbs sampling, then the multivariate sample $(x_1^b, \ldots, x_n^b)$ comes from the joint distribution we aim to simulate.
Without loss of generality, we will prove the next results in the 2-variable case, as it allows for more straightforward computations.

2.5.1 Detailed balance

As already mentioned, the Gibbs sampler fulfills the detailed balance condition. This means that for any given state of the chain,considering a bidimensional example, it holds true that

$$T(\mathbf{x}_{b-1}|\mathbf{x}_b)f(\mathbf{x}_{b-1}) = T(\mathbf{x}_b|\mathbf{x}_{b-1})f(\mathbf{x}_b)$$

Proof. It is sufficient to observe that the transition

$$(x_1^{b-1}, x_2^{b-1}) \to (x_1^b, x_2^b)$$

can be broken into

$$(x_1^{b-1}, x_2^{b-1}) \to (x_1^b, x_2^{b-1}) \to (x_1^b, x_2^b)$$

to conclude that the transition probabilities decompose as follows

$$T(x_1^b, x_2^b|x_1^{b-1}, x_2^{b-1}) = T(x_1^b, x_2^{b-1}|x_1^{b-1}, x_2^{b-1})T(x_1^b, x_2^b|x_1^b, x_2^{b-1})$$

Then let

$$T(x_1^b, x_2^{b-1}|x_1^{b-1}, x_2^{b-1}) = f(x_1^b|x_2^{b-1})$$

that is the full conditional of x_1. It is now trivial to verify the statement

$$\begin{aligned}
&T(x_1^b, x_2^{b-1}|x_1^{b-1}, x_2^{b-1})f(x_1^{b-1}, x_2^{b-1}) = \\
&T(x_1^b, x_2^{b-1}|x_1^{b-1}, x_2^{b-1})T(x_1^b, x_2^b|x_1^b, x_2^{b-1})f(x_1^{b-1}|x_2^{b-1})f(x_2^{b-1}) = \\
&f(x_1^b|x_2^{b-1})f(x_1^{b-1}|x_2^{b-1})f(x_2^{b-1}) = f(x_1^b, x_2^{b-1})T(x_1^{b-1}, x_2^{b-1}|x_1^b, x_2^{b-1})
\end{aligned}$$

By similar arguments, one can prove that

$$T(x_1^b, x_2^{b-1}|x_1^b, x_2^{b-1})f(x_1^b, x_2^{b-1}) = f(x_1^b, x_2^b)T(x_1^b, x_2^{b-1}|x_1^b, x_2^b)$$

The result can be generalized to higher dimension, let's say n, by breaking the transition into n transitions and then applying the above scheme. □

2.6 Metropolis-Hastings algorithm

The Gibbs sampler represents the most straightforward way to sample from a multivariate distribution, provided that we are able to sample from the full conditionals. Unfortunately, this is not quite often the case.
Sometimes we are required to sample from a distribution knowing its kernel (i.e. the pdf without the normalization constant), but we have no information

on the normalization constant or it is simply to expensive to compute.
As a result, a Gibbs algorithm cannot be implemented.
One might argue that using accept-reject methods could solve the problem, but as the dimension increases it becomes, in general, more and more difficult to accept samples as the hyper volume outside the kernel increases as the dimension of the problem increases.
To solve this kind of problems, another well-known algorithm is proposed: the Metropolis-Hastings algorithm.
The idea is to propose a new sample from a known distribution (that must share the same support of the target distribution) and find a way to either accept or not this new sample. In this sense, Metropolis-Hastings algorithm might be seen as an accept-reject method.
Here follows the sampling scheme

Algorithm 2 Metropolis-Hastings sampler

$b = 0$
while b<B **do**
  $b \leftarrow b + 1$
  $x^* \sim q(x^*|x^{b-1})$
  $\alpha \leftarrow \min\left\{\frac{f(x^*)q(x^{b-1}|^*)}{f(x^{b-1})q(x^*|x^{b-1})}, 1\right\}$
  $u \sim U(0,1)$
  if $u \leq \alpha$ **then**
    $x^b \leftarrow x^*$
  else
    $x^b \leftarrow x^{b-1}$
  end if
end while
return $\boldsymbol{X} = (\boldsymbol{x^1}, \ldots, \boldsymbol{x^B})$

This sampling scheme is usually included within a Gibbs sampling scheme, allowing to use the full conditionals whenever they're known and using the algorithm just for those variables that really need it.

2.6.1 Detailed balance

As for the Gibbs sampler, also the Metropolis-Hastings algorithm satisfies the detailed balance.
The transition probability from $(x_1^{b-1}, x_2^{b-1}) \to (x_1^b, x_2^{b-1})$ is then chose as

$$T(x_1^b, x_2^{b-1}|x_1^{b-1}, x_2^{b-1}) = q(x_1^b|x_1^{b-1}, x_2^{b-1})\alpha(x_1^b, x_1^{b-1})$$

with

$$\alpha(x_1^b, x_1^{b-1}) = \min\left\{\frac{f(x_1^b|x_2^{b-1})q(x_1^{b-1}|\boldsymbol{x}^b)}{f(x_1^{b-1}|x_2^{b-1})q(x_1^b|\boldsymbol{x}^{b-1})}, 1\right\}$$

known as *acceptance ratio*. Then, under these assumptions, it holds again that

$$T(x_1^b, x_2^{b-1}|x_1^{b-1}, x_2^{b-1})f(x_1^{b-1}, x_2^{b-1}) = f(x_1^b, x_2^{b-1})T(x_1^{b-1}, x_2^{b-1}|x_1^b, x_2^{b-1})$$

$$T(x_1^b, x_2^{b-1}|x_1^b, x_2^{b-1})f(x_1^b, x_2^{b-1}) = f(x_1^b, x_2^b)T(x_1^b, x_2^{b-1}|x_1^b, x_2^b)$$

Proof. We aim at proving that choosing the metropolis ration as stated, will result in the fulfillment of the detailed balance. First of all, let's write the detailed balance by explicitly writing the transition probabilities, as a function of α:

$$f(x_1^{b-1}, x_2^{b-1})q(x_1^b|x_1^{b-1}, x_2^{b-1})\alpha(x_1^b, x_1^{b-1}) = f(x_1^b, x_2^{b-1})q(x_1^{b-1}|x_1^b, x_2^{b-1})\alpha(x_1^{b-1}, x_1^b)$$

Which can again be rewritten as

$$\frac{\alpha(x_1^b, x_1^{b-1})}{\alpha(x_1^{b-1}, x_1^b)} = \frac{f(x_1^b, x_2^{b-1})q(x_1^{b-1}|x_1^b, x_2^{b-1})\alpha(x_1^{b-1}, x_1^b)}{f(x_1^{b-1}, x_2^{b-1})q(x_1^b|x_1^{b-1}, x_2^{b-1})\alpha(x_1^b, x_1^{b-1})}$$

By Bayes theorem the joint distribution cam be decomposed as following

$$\frac{\alpha(x_1^b, x_1^{b-1})}{\alpha(x_1^{b-1}, x_1^b)} = \frac{f(x_1^b|x_2^{b-1})f(x_2^{b-1})q(x_1^{b-1}|x_1^b, x_2^{b-1})\alpha(x_1^{b-1}, x_1^b)}{f(x_1^{b-1}|x_2^{b-1})f(x_2^{b-1})q(x_1^b|x_1^{b-1}, x_2^{b-1})\alpha(x_1^b, x_1^{b-1})}$$

Leading to this final expression.

$$\frac{\alpha(x_1^b, x_1^{b-1})}{\alpha(x_1^{b-1}, x_1^b)} = \frac{f(x_1^b|x_2^{b-1})q(x_1^{b-1}|x_1^b, x_2^{b-1})\alpha(x_1^{b-1}, x_1^b)}{f(x_1^{b-1}|x_2^{b-1})q(x_1^b|x_1^{b-1}, x_2^{b-1})\alpha(x_1^b, x_1^{b-1})}$$

Now it is sufficient to observe that for sure one between $\alpha(x_1^b, x_1^{b-1})$ and $\alpha(x_1^{b-1}, x_1^b)$ will be equal to one (there are only 2 options: accept the new proposed sample or keep the last one), therefore the just written ratio will always be constant.
More precisely, if $\alpha(x_1^{b-1}, x_1^b) =$ then

$$\frac{\alpha(x_1^b, x_1^{b-1})}{\alpha(x_1^{b-1}, x_1^b)} = \frac{f(x_1^b|x_2^{b-1})q(x_1^{b-1}|x_1^b, x_2^{b-1})\alpha(x_1^{b-1}, x_1^b)}{f(x_1^{b-1}|x_2^{b-1})q(x_1^b|x_1^{b-1}, x_2^{b-1})\alpha(x_1^b, x_1^{b-1})}$$

If otherwise $\alpha(x_1^{b-1}, x_1^b) = 1$ we have

$$\frac{\alpha(x_1^{b-1}, x_1^b)}{\alpha(x_1^b, x_1^{b-1})} = \frac{f(x_1^{b-1}|x_2^{b-1})q(x_1^b|x_1^{b-1}, x_2^{b-1})\alpha(x_1^b, x_1^{b-1})}{f(x_1^b|x_2^{b-1})q(x_1^{b-1}|x_1^b, x_2^{b-1})\alpha(x_1^{b-1}, x_1^b)}$$

As a final consequence, this proves that under this choice of α the detailed balance is satisfied. □

An important point when implementing Metropolis-Hastings algorithm is the choice of the proposal distribution.
As matter of fact, it is preferred a symmetric choice of $q()$ when it is possible. The reason is pretty simple: if $q()$ is symmetric, then

$$q(x^*, x^{b-1}) = q(x^{b-1}, x^*)$$

and under this assumption, the metropolis ratio simplifies as follows

$$\alpha(x_1^b, x_1^{b-1}) = \min\left\{\frac{f(x_1^b|x_2^{b-1})}{f(x_1^{b-1}|x_2^{b-1})}, 1\right\}$$

2.6.2 Link between Gibbs and Metropolis

Before moving on, it is important to notice that Gibbs and Metropolis share some common points. As a matter of fact, the Metropolis algorithm is nothing more then a generalization of the Gibbs sampler.
Let's consider a Metropolis-Hastings sampler and choose the full conditional density as proposal. Then the acceptance ratio becomes

$$\alpha(x_1^b, x_1^{b-1}) = \min\left\{\frac{f(x_1^b|x_2^{b-1})f(x_1^{b-1}|x_2^{b-1})}{f(x_1^{b-1}|x_2^{b-1})f(x_1^b|x_1^{b-1})}, 1\right\} = 1$$

In other words, the Gibbs sampler is a particular case of the Metropolis Hastings one, where the acceptance ratio is always one.
This is particularly important because underlines the computational aspect of choosing the Gibbs sampler whenever it is possible: having an acceptance ratio equal to one means that each iteration will produce a new sample. On the other hand, the Metropolis will generate a new sample in a proportion α of the iterations made, but allows for a more flexible sampling scheme.
In both cases, we end up with a sample of B **dependent** samples, that might be further analyzed to extract a iid subsample.

Chapter 3

Shape and size modelling

3.1 Geometrical concept of shape

In order to make inference on the shape and size space, the latter must be first defined. According to [10] the shape of an object is "everything that remains when translation, rotation and scale information is removed". In order to make proper shape inference, it is needed to first model each of these information and then find a way of "removing" them, leaving us with a shape only configuration.
We will first discuss what translation is, moving then to translation and finally to size-and-shape.

3.2 Rotations

Definition of rotation The concept of rotation is well defined using orthogonal matrices.
As a matter of fact, given a configuration $X \subseteq \mathbb{R}^{K,p}$ of points in space, a rotation can be obtained by pre or post multiplying the matrix X by a matrix $\Gamma \in SO(p)$, where $SO(p)$ is the special orthogonal group of matrices in $\mathbb{R}^p$.
In mathematical words, this means that Γ needs to satisfy

$$\Gamma\Gamma^T = I_p$$
$$det(\Gamma) = 1$$

the first condition is satisfied by all orthogonal matrices, while the second condition is used to characterize the special orthogonal group.
While the definition of rotation is pretty straightforward, it is much less easy to represent them in a convenient way. In the following, we present three methods that might be used to properly represent matrices in some special

cases, i.e. with $p \in \{2, 3\}$.

2D rotations Starting with the $p = 2$ scenario, let's suppose that we want to rotate a vector $[\mathbf{X}, \mathbf{Y}]^T$ of an angle $\theta \in [0, 2\pi)$, obtaining a new vector $[\mathbf{X}, \mathbf{Y}]^T$ By means of simple trigonometric computations, one can easily recover that

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = R\mathbf{X}$$

The matrix $\boldsymbol{R}$ satisfies both the previous conditions, therefore is a rotation matrix itself.
More precisely, $\mathbf{R}$ represents a planar rotation, meaning that the new configuration lies on th same plane of the starting one. Rotations matrices has many interesting properties, ranging from geometrical to numerical ones. From a numerical point of view, rotations matrices do not propagate numerical errors, as they are orthogonal matrices that do not alter the norm of vectors.
As a consequence, using these matrices will be particularly useful even from a practical implementation point of view.
This kind of representation of a rotation can be extended to the 3D case in many ways such as using Euler angles(which decomposes a 3D rotation into a sequence of 3 planar 2D rotations) axis angle representation and quaternions representation. We will focus on the first one as it will allow for a straightforward derivation of the full conditional in the MCMC setting.

3.2.1 Euler angles

As discussed in [11], Euler angles can be defined in different ways. In the following, we will use only the standard Euler angles definition. To define those angles, we first need to define an important axis:

Definition 3.2.1. Node axis
Given an orthonormal basis $\{i, j, k\}$ of a mobile reference system, and a orthonormal basis $\{e_1, e_2, e_3\}$ for a fixed reference system, we define the node axis as the versor

$$\mathbf{n} = \frac{k \times e_3}{|k \times e_3|}$$

Given the node axis, it is possible to define the three angles

Definition 3.2.2. Euler angles

- θ_1 : the angle to rotate $\mathbf{i}$ in the plane orthogonal to $\mathbf{k}$, to obtain the axis node $\mathbf{n}$
- θ_2 : the angle between $\mathbf{e}_3$ and $\mathbf{k}$
- θ_3 the angle to rotate $\mathbf{n}$ in the plane orthogonal to $\mathbf{e}_3$ counterclockwise, in order to obtain $\mathbf{e}_1$

It is possible to prove that, given any fixed reference system $\{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$ and a reference system $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$,a unique value of the three euler angles exist.

Euler angles can then be used to decompose a generic rotation matrix $\boldsymbol{R} \in SO(3)$ into a product of three matrices,i.e.

$$\boldsymbol{R} = \boldsymbol{R}_3(\theta_3)\boldsymbol{R}_2(\theta_2)\boldsymbol{R}_1(\theta_1) =$$

$$\begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 \\ \sin(\theta_3) & \cos(\theta_3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_2) & -\sin(\theta_2) \\ 0 & \sin(\theta_2) & \cos(\theta_2) \end{bmatrix} \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In other words, this means that every 3D rotation can be decomposed into three planar rotations, and as we said this decomposition is unique, provided that the angles satisfy some constraints, i.e. $\theta_1 \in [0, 2\pi], \theta_2 \in [0, \pi), \theta_3 \in [0, 2\pi)$.
Using Euler angles is pretty simple and allow us to work with planar rotations only. Of course, there are some drawbacks.
As a matter of fact, using Euler angles might be computationally expensive: one could store just the angles and compute every time the matrices when needed or, in a sort of opposite way, one could store the 3 matrices and the angle. This choice depends both on the available resources and the context of application. Indeed, this are irrelevant drawbacks when computations can be executed, for instance, on a GPU.

3.3 Translation

The next important geometric concept that needs to be formalized is translation. The idea is to remove the location information by using a linear transformation.
This can be done by multiplying the configuration space by the so called Helmert sub-matrix H, defined in the following way

Definition 3.3.1. Helmert submatrix
The Helmert submatrix $\boldsymbol{H} \in \mathbb{R}^{K,p}$ is such that its $j-th$ row is equal to

$$(-h_j, \ldots, -h_j, jh_j, 0, \ldots, 0)$$

where $-h_j = -1/\sqrt{j(j+1)}$ is repeated j times, followed by jh_j, and then followed by all zeros.

The reason behind this choice is pretty straightforward: any translation of a given configuration $\mathbf{X}_0 \in \mathbb{R}^{K+1,p}$ can be written as

$$\mathbf{X} = \mathbf{X}_0 + \mathbf{1}\mathbf{v}^T \tag{3.1}$$

where $\mathbf{1}^T$ is the column vector with all entries equal to one. Pre multiplying by $\mathbf{H}$ will result in

$$\mathbf{HX} = \mathbf{HX}_0 + \mathbf{H1}\mathbf{v}^T = \mathbf{HX}_0 \tag{3.2}$$

where the last equality follows from the fact that the rows of $\mathbf{H}$ sums to zero. In this way we are removing translation information, but losing a degree of freedom.

The configuration $\mathbf{HX} \in \mathbb{R}^{K\times p}$ obtained by multiplying the landmark configuration by the Helmert sub-matrix is often referred as *helmertised* configuration or *pre-form* matrix.
Using the Helmertized matrix instead of just centering the data, allows to look at the contrasts between landmarks, which is really useful when dealing with shapes.

3.4 Scale

We need to introduce a geometrical framework to isolate the scale information of a given set of landmarks $\mathbf{X}$.
In the following we will briefly explore the possibility of using the SVD decomposition to isolate the size-and-shape of and object, i.e. the confoguration retaining both scale and shape information.

3.4.1 Singular values decomposition - SVD

As already mentioned, the SVD is one of the main ingredients to fully understand and isolate the size-and-shape information. An explicit discussion of SVD can be found in [2], and in the following only the useful result will

be reported. Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m,n}$, the singular values decomposition of A can be defined as

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{R}^T$$

with

$$U \in O(m) \quad \Delta \in \mathbb{R}^{m,n} \quad V \in O(n)$$

SVD decomposition is nothing but a generalized version of the eigenvector problem. As a matter of fact, it holds true that

- U contains the orthonormal eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$
- R contains the orthonormal eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$

The matrix

$$\boldsymbol{\Delta} = \begin{bmatrix} \delta_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \delta_2 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & 0 & 0 \\ 0 & 0 & 0 & \delta_r & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

contains on its diagonal the singular values of the matrix $\boldsymbol{A}$, i.e. the first r diagonal terms, with r being the rank of matrix $\boldsymbol{A}$. These values are related to the previously cited eigenvector problems in the following way: first we solve the eigenvector problem for A^TA

$$\boldsymbol{A}\boldsymbol{A}^T\boldsymbol{v}_k = \delta_k^2\boldsymbol{v}_k \quad k = 1, \ldots, r$$

And then we can retrieve the first r eigenvectors of AA^T

$$\boldsymbol{u}_k = \frac{\boldsymbol{A}\boldsymbol{v}_k}{\delta_k} \quad k =, 1 \ldots, r$$

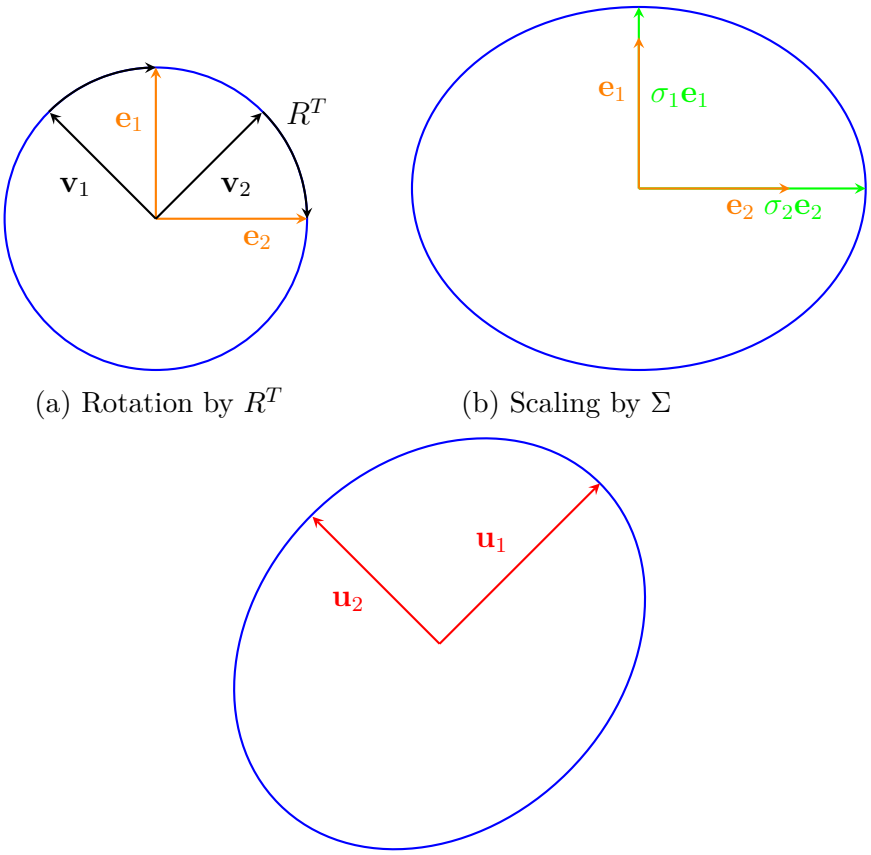
The remaining $n-r$ eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$, and $m-r$ eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$ can be arbitrarily specified and constitutes a basis for the kernel of, respectively, bmA and A^T.
By this arguments, it is easy to verify that the SVD of a matrix is **not unique** and might depend on both the choice of these last eigenvectors, and permutations of the columns of $\boldsymbol{\Delta}$ and $\boldsymbol{R}$.

3.4.2 Geometrical interpretation of SVD

The key point in using the SVD to asses the size-and-shape of an object can be easily understood by looking at its geometrical interpretation.
SVD essentially breaks the action of a given matrix $\boldsymbol{A}$ in three distinct pieces

- $\boldsymbol{R}^T \in O(n)$ is an orthogonal matrix that contains only information about the orientation of the set A.
- $\boldsymbol{\Delta}$ is a matrix that contains information about the scale: its entries, which are the singular values of matrix $\boldsymbol{A}$, define the "stretches" that are performed along each direction.
- $\boldsymbol{U} \in O(m)$ is a rotation matrix that map the stretched configuration to the original orientation.

The idea is that any given set of vectors, under the action of $\boldsymbol{A}$, can be mapped to a reference configuration, stretched only on the principal directions of $\boldsymbol{A}$ and then rotated back to obtain a new set of vectors.



(a) Rotation by R^T

(b) Scaling by Σ

(c) Counter-rotation by U

Figure 3.1: Geometric interpretation of SVD

It is worth noting that the SVD decomposition depends on the space position, in the sense that matrices $\boldsymbol{A}$ and $\boldsymbol{A} + c\mathbf{1}^T\mathbf{1}$ produces different maps and have different singular values deocmposition.
As a consequence, SVD should be applied only after the Helmert sub-matrix previously defined.
Back to the original problem, it follows as a natural consequence that given a helmertised configuration $\boldsymbol{X}_H =, HX$, one can use SVD decomposition to recover the shape and size information, given by the product $\boldsymbol{U\Delta}$ and the rotation information, given by the matrix $\boldsymbol{R}$.

Observation 1. *The SVD decomposition is usually defined using $\boldsymbol{R} \in O(n)$. It is worth noting that it is possible to rewrite such decomposition in a way that $\boldsymbol{R} \in SO(n)$. This can be achieved by changing the sign of a column of*

$\boldsymbol{U}$ and the respective column of $\boldsymbol{R}$, whenever $\boldsymbol{R}$ has determinant equal to -1. In this way reflection information is lost, and therefore the left-handed and right-handed reference systems are considered the same.

Proof. Consider the starting decomposition $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{R}^T$. Changing a column sign, e.g. the first, corresponds to post multiplying by a matrix

$$\boldsymbol{P} = \begin{bmatrix} -1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

if $det(\boldsymbol{R}) = -1$ then it holds by Binet theorem that

$$\det(\boldsymbol{P}\boldsymbol{R}) = \det(\boldsymbol{P})\det(\boldsymbol{R}) = -1\det(\boldsymbol{R}) = 1$$

and

$$\boldsymbol{U}\boldsymbol{P}\boldsymbol{\Delta}(\boldsymbol{R}\boldsymbol{P})^T = \boldsymbol{U}\boldsymbol{P}\boldsymbol{\Delta}\boldsymbol{P}^T\boldsymbol{R}^T = \boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{P}\boldsymbol{P}^T\boldsymbol{R}^T = \boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{R}^T = \boldsymbol{A}$$

which concludes the proof □

3.5 Measuring distances in the shape space

We will now briefly discuss the main properties of the shape space. This will be useful to better understand how one can compare the shape of two different configurations, which will result particularly useful when in the next chapters we will compare estimated configurations with real ones.
The main issue with size-and-shape is that it presents as a non euclidean metric space structure [10], therefore some care must be taken when making comparisons between shapes.

3.5.1 Size-and-shape space

As it will be the study case scenario for the majority of applications considered, we now state the formal definition of what the size-and-shape space is

Definition 3.5.1. The **size-and-shape** of a configuration matrix $\mathbf{X}$ is all the geometrical information about $\mathbf{X}$ that is invariant under location and rotation, i.e. the set

$$[\boldsymbol{X}]_S = \{\boldsymbol{X}_H\boldsymbol{\Gamma} : \boldsymbol{\Gamma} \in SO(p)\}$$

with $\boldsymbol{X}_H$ being the helmertised configuration previously defined

As a direct consequence, the size-and-shape space is defined as

Definition 3.5.2. The **size-and-shape space** is the space of all size-and-shapes.

Size and shape is also known as **form**, especially in biological applications, as it naturally encodes the everyday concept of "form" itself. By allowing $\Gamma \in O(p)$, we refer to **reflection size-and-shape space**. This is particularly important as we might or might not need reflection invariance, depending on the application.

3.5.2 Shape space

By similar arguments the shape space can be defined from the the size-and-shape one, by removing the scale information, leading to the following definition

$$[\boldsymbol{X}] = \{\boldsymbol{Z}\boldsymbol{\Gamma} \quad : \boldsymbol{\Gamma} \in SO(p) \quad \} \tag{3.3}$$

Where $\boldsymbol{Z} := \frac{\boldsymbol{HX}}{\|\boldsymbol{HX}\|}$ is also known as *pre-shape* configuration of $\boldsymbol{X}$.
The choice of working with one space rather than the other clearly depends on the kind of application one is considering. In the following, we will often interchange the use of one space with the other: it might be convenient to make inference on $\boldsymbol{Y}$ and then, only in a second moment, compare the difference between shapes.In this sense, the use of these spaces should be seen as complementary, rather than mutually exclusive.

3.5.3 The partial procrustes distance

The first distance that can be defined between size-and-shape objects is the so called **partial procrustes distance**.
The idea is pretty simple: being the size-and.shape the set of all pre-form differ only by a rotation, one might define the distance between two size-and-shape configurations by minimizing over rotations, i.e.

$$d_p(\boldsymbol{X}_1, \boldsymbol{X_2}) = \inf_{\{\boldsymbol{\Gamma} \in SO(p)\}} \|\boldsymbol{Z}_2 - \boldsymbol{Z}_1\boldsymbol{\Gamma}_1\| \tag{3.4}$$

with $\boldsymbol{Z}_i = \frac{\boldsymbol{HX}_i}{\|\boldsymbol{HX}_i\|}$ Even though this definition is mathematically rigorous, it is not operative. Therefore, this distance can be further exploited

$$d_p(\boldsymbol{X}_1, \boldsymbol{X_2}) = \sqrt{2}\left(1 - \sum_{i=1}^{m} \lambda_i\right)^{\frac{1}{2}} \tag{3.5}$$

where $\lambda_i \quad i = 1, \ldots, m$ are the square roots of the eigenvalues of $\boldsymbol{Z}_1^T \boldsymbol{Z}_2 \boldsymbol{Z}_2^T \boldsymbol{Z}_1$, ordered in a descending way.

3.5.4 The full procrustes distance

Another possible metric that can be used to measure distances between shapes is the **full procrustes distance**. This distance is strictly related to its partial version, with the only difference in minimization: here the minimization is not carried over just by rotations, but also by a factor $\beta \in \mathbb{R}^+$, i.e.

$$d_F(\boldsymbol{X}_1, \boldsymbol{X_2}) = \inf_{\{\boldsymbol{\Gamma} \in SO(p)\ ,\ \beta \in \mathbb{R}^+\}} \|\boldsymbol{Z}_2 - \beta \boldsymbol{Z}_1 \boldsymbol{\Gamma}_1\| \tag{3.6}$$

And again this distance can be characterized in a more operative way

$$d_F(\boldsymbol{X}_1, \boldsymbol{X_2}) = \left(1 - \left(\sum_{i=1}^{m} \lambda_i\right)^2\right)^{\frac{1}{2}} \tag{3.7}$$

with $\lambda_i \quad i = 1, \ldots, m$ defined as in the partial procrustes distance.

3.5.5 Riemannian distance

A third, possibly mor euseful, distance that can be defined to measure distances between shapes is the **Riemannian distance**.
Following the explanations in [10], it emerges that the Riemannian distance is nothing more than the closest great circle distance on the *pre-shape sphere*.
From a mathematical point of view, the distance is defined as

$$\rho(\boldsymbol{X}_1, \boldsymbol{X}_2) = \arccos\left(\sum_{i=1}^{m} \lambda_i\right) \tag{3.8}$$

with the eigenvalues λ_i defined as usual.
The Riemannian distance will be largely used in the following, as it allows to accurate comparisons between shapes.

3.5.6 Comparing the distances

The three just described distances can be compared from both a geometrical and an algebraic point of view.
First of all, it is useful to report the bounds for each of the distances

Distance	Domain
d_P	$[0, \sqrt{2}]$
d_F	$[0, 1]$
ρ	$\left[0, \frac{\pi}{2}\right]$

As can be seen from the above table, the distances don't differ much in range. The real differences and correlations between these amthematical objects can be retrievered by algebraic end geometric means.
From an algebraic point of view, it can be showed that

$$d_F(\boldsymbol{X}_1, \boldsymbol{X}_2) = \sin(\rho)$$

$$d_P(\boldsymbol{X}_1, \boldsymbol{X}_2) = 2\sin(\frac{\rho}{2})$$

while from a geometrical point of view, the three distances ρ, P, F are, respectively, the smallest angle on the *pre-shape sphere section* between the two pre shapes configurations, the chordal length and the "point-line" distance between the second configuration and the radius of the first one.

Chapter 4

Bayesian size-and-shape regression modelling

In the next chapter, we will move into the core of this work.
We aim to define a regressive type relation between the size-and-shape configuration of a sample and some covariates.
In the first section of the chapter, we will briefly recall a stochastic model for size-and-shape
In the second part of the chapter, we will setup the Bayesian framework by defining the priors and deriving the posterior distributions for the parameters, discussing possible identification issues. It is essential to underline all of these topics have already been explored in literature in both the 2 and 3 dimensional cases, as discussed in [10] and [3], therefore we will be just reviewing those contents for the majority of this work.
Nonetheless, a slightly different approach from the original work is here proposed: the Euler angles' full conditionals are derived for the $p = 3$ case, avoiding the metropolis step that is cited in both the articles.

4.1 A stochastic model for size and shape

As already seen, statistical shape analysis may come of use in many cases. The common points for each and every problem of interest is the structure of the dataset.
When dealing with "shapes" datasets,we are given a set of N configurations in space, each of them containing $K + 1$ landmarks

$$\boldsymbol{X}_i \in \mathbb{R}^{K+1\times p} \; i \in \{1, \dots, N\}$$

We generally assume the independence between observations

$$\boldsymbol{X}_i \perp\!\!\!\perp \boldsymbol{X}_j \; i \neq j$$

and we also assume that the components of the landmarks are independent (but we assume no independence between landmarks)

$$\boldsymbol{X}_{i,l} \perp\!\!\!\perp \boldsymbol{X}_{i,k} \; l \neq k$$

The inference can now be done in different ways, as there are many possible assumptions of invariance for the model: one might ask for translation invariance, rotation invariance, reflection invariance and scale invariance by applying one or many of the transformations discussed in previous chapters. In the following, we will assume translation invariance and reflection invariance, meaning that we will only work with the size-and-shape configuration of the object, assuming that information about reflection is lost. First of all, we need to remove translation and this can be done by pre-multiplying by the Helmertz submatrix of order K.
This will lead to a *pre-form* configuration

$$\boldsymbol{X}_H := \boldsymbol{H}\boldsymbol{X} \in \mathbb{R}^{K,p}$$

that could also be standardized if needed.
Once location information is removed, we can retrieve the SVD of $\boldsymbol{X_H}$

$$\boldsymbol{X}_{H,i} = \boldsymbol{U}_i \boldsymbol{\Delta}_i \boldsymbol{V}_i^T$$

In this way the matrix

$$\boldsymbol{Y}_i := \boldsymbol{U}_i \boldsymbol{\Delta}_i$$

will retain shape-and-size information, while the matrix $\boldsymbol{V}_i$ will contain only rotational information.
To be sure that the inference is based only on the size-and-shape information $\boldsymbol{Y}_i$, we replace $\boldsymbol{V}_i$ with a new matrix $\boldsymbol{R}_i \in SO(p)$, i.e.

$$\boldsymbol{X}_i = \boldsymbol{Y}_i \boldsymbol{R}_i^T$$

and then define a regressive relation between this new configuration and a vector of covariates $\boldsymbol{z}_i$ in the following way

$$\boldsymbol{X}_{H,i} \sim \mathcal{N}_{k,p}\left(\sum_{h=1}^{d} z_{ih} \boldsymbol{B_h}, \boldsymbol{I}_p \otimes \boldsymbol{\Sigma} \right) \; i = 1, \ldots, N$$

where

- $\boldsymbol{z}_i \in \mathbb{R}^{N,d}$ is a vector of covariates for the $i-th$ sample
- $\boldsymbol{B}_h \in \mathbb{R}^{K,p}$ $h = 1, \ldots, d$ are matrices of regressive coefficients
- $\boldsymbol{\Sigma} \in \mathbb{R}^{K,K}$ is a variance-covariance matrix

A crucial point is to observe that, in the following, we will never use the real rotations $\boldsymbol{V}_i$. Moreover, by assuming that both $\boldsymbol{V}_i$ and $\boldsymbol{R}_i$ are special orthogonal matrices, we are retaining the reflection information.

4.2 The bayesian framework

After a stochastic model for size-and-shape is set up, the next step is to derive a Bayesian model to estimate the parameters.
Working in a Bayesian setting, we assume that the parameters are random variables themselves. Before observing data, we might have some beliefs about these parameters, and therefore we exploit our prior knowledge by defining the prior distributions on the parameters. These priors will then be driven by empirical observation by applying the well-known Bayes update rule. Before moving on, some notation must be clarified. Let

$$\boldsymbol{\beta_l} = [\boldsymbol{B}_{1,l}^T, \ldots, \boldsymbol{B}_{d,l}^T]^T$$

be the vector obtained by stacking the $l-ths$ columns of $\boldsymbol{B}_h$ $h = 1, \ldots, d$

$$\boldsymbol{Z}_i = \boldsymbol{I}_k \otimes \boldsymbol{z}_i^T$$

a design matrix associated to the $i-th$ covariate and

$$\boldsymbol{X}_{i,l}$$

the l-th column of observation i.
We can then define the following priors

$$\begin{aligned}
&\boldsymbol{X}_{i,l}|\boldsymbol{\beta}, \boldsymbol{\Sigma} \sim \mathcal{N}_K(\boldsymbol{Z}_i\boldsymbol{\beta}_l, \boldsymbol{\Sigma}), \quad i = 1, \ldots, N \quad l = 1, \ldots, p \\
&\boldsymbol{\beta}_l \sim \mathcal{N}_{kd}(\boldsymbol{M}_l, \boldsymbol{V}_k) \\
&\Sigma \sim IW(\nu, \boldsymbol{\Psi})
\end{aligned}$$

Keeping in mind all that has been pointed out up to now, we can move to the derivation of the posterior distributions.

4.2.1 Posterior distribution for B

Given the priors, it is easy to prove that the posterior full conditional distribution for $\boldsymbol{\beta_l}$ is given by

$$\boldsymbol{\beta_l}|\boldsymbol{\beta_{-l}}, \Sigma, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_N, \boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_N \sim \mathcal{N}_{kd}(\boldsymbol{M}_l^*, \boldsymbol{V}_l^*)$$

where

$$\boldsymbol{M}_l^* = \boldsymbol{V}_l^* \Bigg(\sum_{i=1}^{N} \boldsymbol{Z_i^T \Sigma^{-} 1 X}_{i,l} + \boldsymbol{V}_l^{-} 1 \boldsymbol{M}_l \Bigg)$$

$$\boldsymbol{V}_l^* \Bigg(\sum_{i=1}^{N} \boldsymbol{Z}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{Z_i} + \boldsymbol{V}_l^{-1} \Bigg)^{-1}$$

Proof. Let $\boldsymbol{\beta}_{-l}$ be the set of all $\boldsymbol{\beta}$ but the $l-th$ one. Then, by Bayes theorem, one retrieve that

$$f(\boldsymbol{\beta_l}|\boldsymbol{\beta_{-l}}, \Sigma, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_N, \boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_N) = f(\boldsymbol{\beta_l}|\boldsymbol{\beta_{-l}}, \Sigma, \boldsymbol{R}, \ldots, \boldsymbol{Y}_N) \propto$$

$$\propto f(\boldsymbol{X}_l|\boldsymbol{\beta_{-l}}, \boldsymbol{\Sigma}, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_N, \boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_N) f(\boldsymbol{\beta_l})$$

Which yields to

$$f(\boldsymbol{\beta_l}|\boldsymbol{\beta_{-l}}, \boldsymbol{\Sigma}, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_N, \boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_N) \propto$$

$$\propto \exp\Big\{ -\frac{1}{2} \big(\boldsymbol{X}_l - \boldsymbol{\mu}\big)^T \boldsymbol{\Sigma}^{-1} \big(\boldsymbol{X}_l - \boldsymbol{\mu}\big) \Big\} \exp\Big\{ -\frac{1}{2} \big(\boldsymbol{\beta}_l - \boldsymbol{M}_l\big)^T \boldsymbol{V}_l^{-1} \big(\boldsymbol{\beta}_l - \boldsymbol{M}_l\big) \Big\}$$

By independence, we can factorize the density of $\boldsymbol{X}$ as follows

$$f(\boldsymbol{X}_l|\boldsymbol{\beta_{-l}}, \boldsymbol{\Sigma}, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_N, \boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_N) =$$

$$= \prod_i f(\boldsymbol{X}_{i,l}|\boldsymbol{\beta_{-l}}, \boldsymbol{\Sigma}, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_N, \boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_N) =$$

$$= \exp\Big\{ -\frac{1}{2} \sum_i \big(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\big)^T \boldsymbol{\Sigma}^{-1} \big(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\big) \Big\}$$

Observing now that

$$\exp\Big\{ -\frac{1}{2} \sum_i \big(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\big)^T \boldsymbol{\Sigma}^{-1} \big(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\big) \Big\} =$$

$$\propto \exp\Big\{ -\frac{1}{2} \sum_i -(\boldsymbol{Z}_i\boldsymbol{\beta}_l)^T \boldsymbol{\Sigma}^{-1} \boldsymbol{X}_{i,l} - \boldsymbol{X}_{i,l}^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{Z}_i\boldsymbol{\beta}_l) + (\boldsymbol{Z}_i\boldsymbol{\beta}_l)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{Z}_i\boldsymbol{\beta}_l) \Big\} =$$

$$= \exp\left\{ -\frac{1}{2}\sum_i -2\boldsymbol{\beta}_l^T \boldsymbol{Z}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{X}_{i,l} + \boldsymbol{\beta}_l^T \boldsymbol{Z}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{Z}_i \boldsymbol{\beta}_l \right\}$$

We can recover the following kernel for the posterior distribution of $\boldsymbol{\beta_l}$

$$f(\boldsymbol{\beta_l}|\boldsymbol{\beta_{-l}}, \boldsymbol{\Sigma}, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_N, \boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_N) \propto$$

$$\propto \exp\left\{ -\frac{1}{2}\sum_i -2\boldsymbol{\beta}_l^T (\boldsymbol{Z}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{X}_{i,l} + \boldsymbol{V}_l^{-1} \boldsymbol{M}_l) + \boldsymbol{\beta}_l^T \boldsymbol{Z}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{Z}_i \boldsymbol{\beta}_l + \boldsymbol{\beta}_l^T \boldsymbol{V}_l^{-1} \boldsymbol{\beta}_l - 2\boldsymbol{\beta}_l^T \boldsymbol{V}_l^{-1} \boldsymbol{M}_l \right\}$$

$$= \exp\left\{ -\frac{1}{2}\sum_i -2\boldsymbol{\beta}_l^T \boldsymbol{Z}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{X}_{i,l} + \boldsymbol{\beta}_l^T (\boldsymbol{Z}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{Z}_i + \boldsymbol{V}_l^{-1}) \boldsymbol{\beta}_l \right\}$$

which is exactly the kernel of a normal distribution with mean $\boldsymbol{M}_l^*$ and variance matrix $\boldsymbol{V}_l^*$ defined as

$$\boldsymbol{M}_l^* = \boldsymbol{V}_l^* \left(\sum_{i=1}^N \boldsymbol{Z_i^T \Sigma^{-1} X}_{i,l} + \boldsymbol{V}_l^{-1} \boldsymbol{M}_l \right)$$

$$\boldsymbol{V}_l^* \left(\sum_{i=1}^N \boldsymbol{Z}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{Z_i} + \boldsymbol{V}_l^{-1} \right)^{-1}$$

□

4.2.2 Posterior for Sigma

Next, we compute the posterior distribution for Σ. The claim is that

$$\boldsymbol{\Sigma}|\boldsymbol{\beta}, \boldsymbol{R}_1, \ldots, \boldsymbol{R_n}, \boldsymbol{Y_1}, \ldots, \boldsymbol{Y_N} \sim IW(\nu^*, \boldsymbol{\Psi}^*)$$

with

$$\nu^* = \nu + Np, \quad \boldsymbol{\Psi}^* = \boldsymbol{\Psi} + \sum_{i=1}^N \sum_{l=1}^p (\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i \boldsymbol{\beta}_l)(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i \boldsymbol{\beta}_l)^T$$

The proof can be derived in similar ways to what was previously done with β_l.

Proof. By Bayes theorem, it is easy to verify that

$$f(\boldsymbol{\Sigma}|\boldsymbol{B}, \boldsymbol{R_1}, \ldots, \boldsymbol{R_n}, \boldsymbol{Y_1}, \ldots, \boldsymbol{Y_N}) \propto f(\boldsymbol{\Sigma}) f(\boldsymbol{X_1}, \ldots, \boldsymbol{X_N}|\boldsymbol{\Sigma}, \boldsymbol{B_1}, \ldots, \boldsymbol{B_d}) =$$

$$= \det(\boldsymbol{\Sigma})^{-\frac{(\nu+K+1)}{2}} \exp\left(-\frac{1}{2} \operatorname{tr}\left(\boldsymbol{\Psi}\boldsymbol{\Sigma}^{-1}\right)\right) \times$$

$$\times \det(\boldsymbol{\Sigma})^{-\frac{np}{2}} \prod_i \prod_l \exp\left(-\frac{1}{2}\left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)^T \boldsymbol{\Sigma}^{-1}\left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)\right) =$$

$$= \det(\boldsymbol{\Sigma})^{-\frac{\nu+np+K+1}{2}} \exp\left(\sum_i \sum_l (\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l)^T \boldsymbol{\Sigma}^{-1}\left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right) + \operatorname{tr}(\boldsymbol{\Psi}\boldsymbol{\Sigma}^{-1})\right)$$

Where we used independence of the $\boldsymbol{X}_i$ and independence of the columns of $\boldsymbol{B}_h$.
By the trace operator properties we have that

$$\operatorname{tr}(\boldsymbol{b}\boldsymbol{a}^T) = \boldsymbol{a}^T\boldsymbol{b} \quad \boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$$

And therefore it holds that

$$\operatorname{tr}\left(\left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)\left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)^T \boldsymbol{\Sigma}^{-1}\right) = \operatorname{tr}\left(\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)\left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)^T\right) =$$

$$= \left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)^T \boldsymbol{\Sigma}^{-1}\left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)$$

This result finally leads to

$$f(\boldsymbol{\Sigma}|\boldsymbol{B}, \boldsymbol{R}_1, \ldots, \boldsymbol{R_n}, \boldsymbol{Y_1}, \ldots, \boldsymbol{Y_N}) \propto$$

$$\propto \det(\boldsymbol{\Sigma})^{-\frac{\nu+np+K+1}{2}} \exp\left(\sum_i \sum_l \operatorname{tr}\left(\left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)\left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)^T \boldsymbol{\Sigma}^{-1}\right) + \operatorname{tr}(\boldsymbol{\Psi}\boldsymbol{\Sigma}^{-1})\right) =$$

$$= \det(\boldsymbol{\Sigma})^{-\frac{\nu+np+K+1}{2}} \exp\left(\operatorname{tr}\left(\left[\sum_i \sum_l \left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)\left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)^T + \boldsymbol{\Psi}\right]\boldsymbol{\Sigma}^{-1}\right)\right)$$

By letting

$$\nu^* := \nu + np \quad \boldsymbol{\Psi}^* := \sum_i \sum_l \left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)\left(\boldsymbol{X}_{i,l} - \boldsymbol{Z}_i\boldsymbol{\beta}_l\right)^T + \boldsymbol{\Psi}$$

we recognize an Inverse Wishart kernel with parameter ν^* and $\boldsymbol{\Psi}^*$, proving the thesis

□

4.2.3 Posterior for R

Concerning the distribution of R_i we already mentioned that

$$f(\boldsymbol{R_i}|\boldsymbol{\beta}, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_{i-1}, \boldsymbol{R}_{i+1}, \ldots, \boldsymbol{R}_N, \boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_N) \propto \exp\left(tr(\boldsymbol{R}_i A_i^T)\right)$$

and again, as mentioned before, we could decompose such matrix by using Euler angles. In this way, we can derive the following posteriors for the angles in the cases $p = 2, 3$.

4.2.4 Euler angles full conditional - case p = 2

Previous works implemented the simulation from a matrix fisher by using a metropolis step. Here we propose a different approach, trying to derive a full conditional of the Euler angles.
In the 2D case, planar rotations can be represented by means of a single angle θ

$$\boldsymbol{R}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix}$$

and in this way, it is easy to characterize the above mentioned kernel

$$\exp(\mathrm{tr}(\boldsymbol{R}_i \boldsymbol{A}^T) = \mathrm{tr}\left(\begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix} \right) =$$

$$= \cos\theta_i a_{11} - \sin\theta_i a_{12} + \sin\theta_i a_{21} + \cos\theta_i a_{22} =$$

$$= \cos\theta_i(a_{11} + a_{22}) + \sin\theta_i(a_{21} - a_{12})$$

Remark that

$$a\cos(x) + b\sin(x) = \sqrt{a^2 + b^2}\left(\frac{a}{\sqrt{a^2 + b^2}}\cos(x) + \frac{b}{\sqrt{a^2 + b^2}}\sin(x) \right) =$$

$$= \rho\cos(\gamma)\cos(x) + \sin(\gamma)\sin(x) = \rho\cos(\gamma - x) = \rho\cos(x - \gamma)$$

And therefore by letting

$$\rho := \sqrt{(a_{11} + a_{22})^2 + (a_{21} - a_{12})^2}$$

$$\gamma := \arctan 2\left(\frac{a_{21} - a_{12}}{\rho}, \frac{a_{11} + a_{22}}{\rho} \right)$$

we finally obtain

$$\cos\theta_i(a_{11} + a_{22}) + \sin\theta_i(a_{21} - a_{12}) = \rho\cos(\theta_i - \gamma)$$

that is the kernel of a Von-mises with parameters ρ and γ

4.2.5 Euler angles full conditional - case p = 3

The derivation in the $p = 3$ case is quite similar: assume an Euler angles decomposition for R_i, using the ZXZ convention. This results in

$$R_i = R_{i3}(\theta_3)R_{i2}(\theta_2)R_{i1}(\theta_1), \quad \theta_1, \theta_3 \in [0, 2\pi), \quad \theta_2 \in [0, \pi)$$

with

$$\boldsymbol{R}_{i1} = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & -\cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{R}_{i2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_2) & -\sin(\theta_2) \\ 0 & \sin(\theta_2) & cos(\theta_2) \end{bmatrix}$$

$$\boldsymbol{R}_{i3} = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 \\ \sin(\theta_3) & \cos(\theta_3) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Following the procedures in [5] we obtain that the invariant volume (Haar) measure of $d\boldsymbol{R}$ over $SO(3)$ can be exploited as a function of the measures of volume of the three angles::

$$d\boldsymbol{R} = \frac{1}{8\pi^2}\sin(\theta_2)d\theta_1 d\theta_2 d\theta_3$$

Last, we can rewrite the density of $\boldsymbol{R_i}$ as

$$\exp(tr(\boldsymbol{R}_i\boldsymbol{A}_i^T)) = \exp(\mathrm{tr}(\boldsymbol{R}_{i3}\boldsymbol{R}_{i2}\boldsymbol{R}_{i1}\boldsymbol{A}_i^T))$$

Observe now that the trace operator is invariant under cyclic permutation (provided that matrices are still compatible) in order to obtain

$$\exp(\mathrm{tr}(\boldsymbol{R}_{i3}\boldsymbol{R}_{i2}\boldsymbol{R}_{i1}\boldsymbol{A}_i^T)) = \exp(\mathrm{tr}(\boldsymbol{R}_{i1}\boldsymbol{A}_i^T\boldsymbol{R}_{i3}\boldsymbol{R}_{i2})) = \exp(\mathrm{tr}(\boldsymbol{R}_{i2}\boldsymbol{R}_{i1}\boldsymbol{A}_i^T\boldsymbol{R}_{i3}))$$

Letting

$$\begin{aligned} L &:= \boldsymbol{R}_{i2}\boldsymbol{R}_{i1}\boldsymbol{A}_i^T \\ H &:= \boldsymbol{A}_i^T\boldsymbol{R}_{i3}\boldsymbol{R}_{i2} \\ D &:= \boldsymbol{R}_{i1}\boldsymbol{A}_i^T\boldsymbol{R}_{i3} \end{aligned}$$

We obtain

$$\exp(\mathrm{tr}(\boldsymbol{R}_{i3}L)) = \exp(\mathrm{tr}(\boldsymbol{R}_{i1}H)) = \exp(\mathrm{tr}(\boldsymbol{R}_{i2}D))$$

to simplify the notation we set

$$c_i := \cos\theta_i \quad s_i := \sin\theta_i \quad i \in 1,2,3$$

It is now possible to explicitly compute the product

$$\mathrm{tr}\left(\boldsymbol{R}_{i1} = \begin{bmatrix} c_1 & -s_1 & 0 \\ s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}\right) =$$

$$= c_1h_{11} - s_1h_{21} + s_1h_{12} - c_1h_{22} + h_{33} = (h_{11} + h_{22})c_1 + (h_{12} - h_{21})s_1 + h_{33} =$$

where we set, in similar way to the $p = 2$ case, the following parameters

$$= \rho_1 \cos(\theta_1 - \gamma_1) + h_{33}$$

$$\rho_1 := \sqrt{(h_{11} + h_{22})^2 + (h_{12} - h_{21})^2}$$

$$\gamma_1 := \arccos\left(\frac{h_{11} + h_{22}}{\rho_1}\right)$$

By similar arguments we can also derive

$$\mathrm{tr}(\boldsymbol{R_{i3}}L) = \rho_3 \cos(\theta_3 - \gamma_3) + l_{33}$$

$$\rho_3 := \sqrt{(l_{11} + l_{22})^2 + (l_{12} - l_{21})^2}$$

$$\gamma_3 := \arccos\left(\frac{l_{11} + l_{22}}{\rho_3}\right)$$

While for θ_2

$$\mathrm{tr}\left(R_{i2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_2 & -s_2 \\ 0 & s_2 & c_2 \end{bmatrix} \begin{bmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{bmatrix}\right) =$$

$$= (d_{22} + d_{33})c_2 + (d_{23} - d_{32})s_2 + d_{11} = \rho_2 \cos(\theta_2 - \gamma_2) + d_{11}$$

And again we set the parameters

$$\rho_2 := \sqrt{(d_{22} + d_{33})^2 + (d_{23} - l_{32})^2}$$

$$\gamma_2 := \arccos\left(\frac{d_{22} + d_{33}}{\rho_2}\right)$$

The following step is to use the decomposition result for $d\boldsymbol{R}_i$ to properly find a kernel of the full conditionals

$$f(\boldsymbol{R_i}|\dots)d\boldsymbol{R_i} = \frac{1}{8\pi^2} f(\theta_{i1}, \theta_{i2}, \theta_{i3}|\dots)\sin(\theta_{i2})d\theta_{i1}d\theta_{i2}d\theta_{i3} = \quad i = 1, \dots, N$$

Finally, to find the full conditional of a specific angle, we just need to isolate the appropriate kernel:

$$f(\theta_{i1}|\dots) \propto \exp(tr(\boldsymbol{R_{i1}}\boldsymbol{H}_i))\sin(\theta_{i2}) \propto \exp\left(\rho_{i1}\cos(\theta_{i1} - \gamma_{i1})\right)$$

$$f(\theta_{i2}|\dots) \propto \exp(tr(\boldsymbol{R_{i2}}\boldsymbol{D}_i))\sin(\theta_{i2}) \propto \exp\left(\rho_{i2}\cos(\theta_{i2} - \gamma_{i2})\right)\sin(\theta_{i2})$$

$$f(\theta_{i3}|\dots) \propto \exp\left(tr(\boldsymbol{R_{i3}}\boldsymbol{L}_i)\right)\sin(\theta_{i3}) \propto \exp\left(\rho_{i3}\cos(\theta_{i3} - \gamma_{i3})\right) i = 1, \dots, N$$

We can finally conclude that the first and the third distributions are Von-Mises.
The second one, instead, is not a known distribution and therefore must be simulated using other techniques, such as accept-reject sampling.

4.3 Sampling the second angle

We just derived a kernel for the distribution of θ_2 and the full conditionals for θ_1 and θ_3. While the latter may be sampled directly from a Von-Mises distribution, the first requires a different approach.
To sample from the distribution of θ_2 we can use an accept-reject methods, that we already cited when discussing the Metropolis-Hastings algorithm. In order to better explain what these methods are, it useful to recall the following

Theorem 3. *Let $f(x)$ be a pdf for a random variable X with finite support. Consider a second random variable $U \sim U(0, 1)$. Then*

$$(X, U) \sim U(A)$$

where

$$A = \{(x, y) : 0 < u < f(x)\}$$

This comes really useful when we want to obtain one or more samples from $f(x)$ but we are not directly able to do it. Here we just need to be able to simulate from a uniform distribution in order to retrieve samples of X. Unfortunately, this might not be necessarily simpler than the original problem.
From a practical point of view, this can be implemented by simulating a

point from a "box" $[a, b] \times [0, m]$ and then retaining only the samples (x, y) that satisfy $y \leq f(x)$.
These kind of approach can be generalized to kernels, as stated in the following theorem

Theorem 4. *Let $X \sim f(x)$ and $Y \sim G$.*
Suppose $\exists M \geq 1$ s.t. $f(x) \leq Mg(x)$.
Then, to obtain a sample from X it is sufficient to sample from $U|Y = y \sim U(0, Mg(y))$ until $0 < u < f(y)$.

This result comes really useful in our scenario, as one can observe that

$$f(\theta_2|\theta_1, \theta_3, \boldsymbol{Y}, \boldsymbol{\mu}) \leq \exp\{\rho \cos(\theta_2 - \gamma_2)\}$$

which is nothing more but the kernel of a Von-Mises distribution.
This means that, at each iteration, it will be sufficient to generate a sample from a Von-mises and accept it if and only if it satisfies the above theorem. This is very handy as it allows to write one single function that is able to sample from both the Von-mises and the new distribution, with little changes in the code.

4.4 Identification of the model

One important fact that has been pointed out by [3] is that the model itself, as is written, suffers of some identification issues: this means that there exist multiple set of the parameters that induces the same likelihood in the model. As a consequence, it is not possible to distinguish one set from the other.
To be more precise, remember that we aim at modeling the size-and-shape by using a linear model

$$\boldsymbol{X}_i|\boldsymbol{z}_i \sim \mathcal{N}\left(\sum_{h=1}^{d} z_{ih}\boldsymbol{B}_h, \boldsymbol{I}_p \otimes \boldsymbol{\Sigma}\right)$$

And we already proved that

$$f(\boldsymbol{Y}|\boldsymbol{\Sigma}, \boldsymbol{\mu}, \boldsymbol{R}) \propto \exp\{-\frac{1}{2}\operatorname{tr}\left[\boldsymbol{\Delta}\boldsymbol{U}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{U}\boldsymbol{\Delta} + \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right]\}$$

Let's focus on the trace: one could define an alternative configuration for parameter $\boldsymbol{\mu}$ by using an orthogonal matrix $\boldsymbol{\Lambda}$, obtaining the same value of the trace operator, i.e.

$$\operatorname{tr}\left((\boldsymbol{\mu}\boldsymbol{\Lambda})^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\boldsymbol{\Lambda}\right) = \operatorname{tr}\left(\boldsymbol{\Lambda}^T\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\boldsymbol{\Lambda}\right) = \operatorname{tr}\left(\boldsymbol{\Lambda}\boldsymbol{\Lambda}^T\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right) = \operatorname{tr}\left(\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right)$$

Where the last equality can be derived by using the fact that the trace is invariant by cyclic permutations (provided the the matrices are compatibles). As a consequence, it is easy to verify that

$$f(\boldsymbol{Y}|\boldsymbol{\Sigma}, \boldsymbol{\mu}, \boldsymbol{R}) = f(\boldsymbol{Y}|\boldsymbol{\Sigma}, \boldsymbol{\mu}\boldsymbol{\Lambda}, \boldsymbol{R})$$

This means that, in order to make any inference, the model parameters needs to be identified. As reported in [4], one possible choice is to define a matrix $\boldsymbol{\Lambda}$ s.t.

$$\boldsymbol{B_{id}} := \boldsymbol{B_0}\boldsymbol{\Lambda}$$

$$[\boldsymbol{B_{id}}]_{ij} = 0 \quad j > i$$

$$[\boldsymbol{B_{id}}]_{ii} \geq 0 \quad i, j = 1 \ldots, p$$

The choice of defining $\boldsymbol{\Lambda}$ using $\boldsymbol{B}_0$ is purely arbitrary and migh therefore be changed. Once $\boldsymbol{\Lambda}$ is defined, each and every $\boldsymbol{B}_h \quad h = 2, \ldots, d$ can be identified by post-multiplying by $\boldsymbol{\Lambda}$. The same identification must be performed also for the rotation matrices: we know that

$$\boldsymbol{Y}\boldsymbol{R}^T = \sum_{h=1}^{d} \boldsymbol{Z}_h\boldsymbol{B}_h + \boldsymbol{\epsilon} \quad \boldsymbol{\epsilon} \sim \mathcal{N}_{K,p}\Big(\boldsymbol{0}_{K,p}, \boldsymbol{I}_p \otimes \boldsymbol{\Sigma}\Big)$$

Therefore

$$\boldsymbol{Y} = \sum_{h=1}^{d} \boldsymbol{Z}_h\boldsymbol{B}_h\boldsymbol{R} + \boldsymbol{\epsilon}\boldsymbol{R}$$

But

$$\boldsymbol{B}_h\boldsymbol{R} = \boldsymbol{B}_h\boldsymbol{\Lambda}\boldsymbol{\Lambda}^T\boldsymbol{R}^T = \boldsymbol{B}_h\boldsymbol{\Lambda}(\boldsymbol{\Lambda}^T\boldsymbol{R})^T$$

for any choiche of $\boldsymbol{\Lambda} \in SO(p)$. Therefore, once the $\boldsymbol{B_h}$ are identified, the corresponding rotation matrix can be retrieved by pre-multiplying by $\boldsymbol{\Lambda}^T$.

Chapter 5

Simulations and experiments

In the first section of this chapter we test the performances of the model by first generating some synthetic datasets and then observing the results.
In the second section, a real dataset application is also proposed.

5.1 p = 2 - only intercept

The first, most simple model that can be discussed is the intercept only model. We generate a sample dataset with $d = 1$ covariates,i.e. the intercept, sampling each of the regressive coefficients from a normal distribution with parameters $\mu = 5$ and $\sigma^2 = 1$.
We also simulate the matrix $\mathbf{\Sigma}$ from a InverseWishart distribution of parameters $\nu = K + 1$ and $\mathbf{\Psi} = 5\boldsymbol{I}_K$.
The reasons behind these choices are quite simple: in a real setting we will of course work with centered and standardized datasets, therefore we will always work with standardized coordinates and covariances, that result in a much more stable sampling. Choosing these parameters allow to test the performances of the sampler in a coherent, yet not trivial setting, showing off both the performances and robustness limits of the sampler.
For each combination of $N \in \{20, 50\}, K \in \{5, 10\}$ we generate 100 datasets and perform MCMC approximation using 3000 iterations, of which 1000 are discarded as burn-in. We then compute:

- The mean length for the 95% credible intervals of each parameter
- The fraction of times that the true value of a certain parameter's component falls into the corresponding 95% credible interval
- The fraction of times that the overall components of a single parameter falls into the corresponding credible interval

- The Riemmanian distance between the estimated configuration and the real one

In the following, we briefly report the just mentioned quantities.

5.1.1 N = 20, K = 10

$B_{0,ij}$	$j=1$	$j=2$
$i=1$	0.94	-
$i=2$	0.95	0.93
$i=3$	0.9	0.97
$i=4$	0.97	0.97
$i=5$	0.97	0.92
$i=6$	0.96	0.95
$i=7$	0.96	0.94
$i=8$	0.95	0.94
$i=9$	0.93	0.96
$i=10$	0.96	0.93

Table 5.1: Fraction of times that true value of $B_{0,ij}$ fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$
$i=1$	0.93	0.93	0.97	0.92	0.97	0.96	0.95	0.94	0.89	0.96
$i=2$	0.93	0.89	0.94	0.92	0.94	0.91	0.93	0.97	0.91	0.9
$i=3$	0.97	0.94	0.94	0.93	0.95	0.93	0.96	0.94	0.93	0.92
$i=4$	0.92	0.92	0.93	0.91	0.93	0.92	0.92	0.97	0.92	0.92
$i=5$	0.97	0.94	0.95	0.93	0.95	0.95	0.91	0.92	0.92	0.95
$i=6$	0.96	0.91	0.93	0.92	0.95	0.94	0.93	0.94	0.93	0.94
$i=7$	0.95	0.93	0.96	0.92	0.91	0.93	0.95	0.93	0.94	0.94
$i=8$	0.94	0.97	0.97	0.92	0.94	0.93	0.95	0.95	0.96	0.94
$i=9$	0.89	0.91	0.93	0.92	0.92	0.93	0.94	0.95	0.94	0.93
$i=10$	0.96	0.9	0.92	0.92	0.95	0.94	0.94	0.96	0.93	0.96

Table 5.2: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	94.7%	1.81
Σ	93.54%	4.13

Table 5.3: Total percentage of times that each parameter fell into the 95% CI

5.1.2 N = 50, K = 10

$B_{0,ij}$	$j=1$	$j=2$
$i=1$	0.89	-
$i=2$	0.94	0.95
$i=3$	0.91	0.95
$i=4$	0.95	0.97
$i=5$	0.92	0.96
$i=6$	0.93	0.97
$i=7$	0.92	0.94
$i=8$	0.91	0.95
$i=9$	0.93	0.95
$i=10$	0.96	0.95

Table 5.4: Fraction of times that true value of $B_{0,ij}$ fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$
$i=1$	0.94	0.96	0.97	0.94	0.92	0.94	0.95	0.92	0.99	0.92
$i=2$	0.96	0.95	0.93	0.93	0.95	0.95	0.92	0.96	0.94	0.93
$i=3$	0.97	0.93	0.92	0.93	0.94	0.93	0.9	0.95	0.93	0.94
$i=4$	0.94	0.93	0.94	0.95	0.95	0.96	0.92	0.95	0.91	0.93
$i=5$	0.92	0.95	0.94	0.95	0.92	0.94	0.94	0.92	0.91	0.90
$i=6$	0.94	0.95	0.93	0.96	0.94	0.94	0.96	0.93	0.96	0.95
$i=7$	0.95	0.92	0.9	0.92	0.94	0.96	0.93	0.91	0.92	0.89
$i=8$	0.92	0.96	0.95	0.95	0.92	0.93	0.91	0.95	0.97	0.92
$i=9$	0.99	0.94	0.93	0.91	0.91	0.96	0.92	0.97	0.94	0.93
$i=10$	0.92	0.93	0.94	0.93	0.9	0.95	0.89	0.92	0.93	0.94

Table 5.5: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	93.95%	1.48
Σ	93.62%	2.42

Table 5.6: Total percentage of times that each parameter fell into the 95% CI

5.1.3 N = 20, K = 5

$B_{0,ij}$	$j = 1$	$j = 2$
$i = 1$	0.95	-
$i = 2$	0.92	0.91
$i = 3$	0.96	0.91
$i = 4$	0.94	0.91
$i = 5$	0.92	0.94

Table 5.7: Fraction of times that true value of $B_{0,ij}$ fell into the 95% CI

Σ_{ij}	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$	0.97	0.93	0.97	0.93	0.88
$i = 2$	0.93	0.93	0.98	0.93	0.87
$i = 3$	0.97	0.98	0.95	0.94	0.96
$i = 4$	0.93	0.93	0.94	0.88	0.91
$i = 5$	0.88	0.87	0.96	0.91	0.91

Table 5.8: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	92.89%	1.73
Σ	92.96%	2.03

Table 5.9: Total percentage of times that each parameter fell into the 95% CI

5.1.4 N = 50, K = 5

$B_{0,ij}$	$j=1$	$j=2$
$i=1$	0.96	-
$i=2$	0.95	0.94
$i=3$	0.97	0.92
$i=4$	0.94	0.94
$i=5$	0.91	0.90

Table 5.10: Fraction of times that true value of $B_{0,ij}$ fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	0.95	0.98	0.96	0.92	0.94
$i=2$	0.98	0.99	0.95	0.94	0.95
$i=3$	0.96	0.95	0.98	0.97	0.95
$i=4$	0.92	0.94	0.97	0.93	0.97
$i=5$	0.94	0.95	0.95	0.97	0.93

Table 5.11: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	93.67%	1.01
Σ	95.36%	1.32

Table 5.12: Total percentage of times that each parameter fell into the 95% CI

Configuration	Riemmanian distance ρ
N= 20, K = 10	0.056
N = 50, K = 10	0.044
N= 20, K = 5	0.039
N = 50, K = 5	0.017

Table 5.13: Comparing Riemmanian distances

5.2 p = 2 - normal covariate

In the following, we report the results for the normal covariate case.
We suppose that each $z_i \sim \mathcal{N}(0,1)$, as standardization will always be per-

formed (the same approach will be used for the $p = 3$ case).

5.2.1 N = 20, K = 10

$B_{0,ij}$	$j = 1$	$j = 2$
$i = 1$	0.95	-
$i = 2$	0.93	0.95
$i = 3$	0.94	0.93
$i = 4$	0.93	0.92
$i = 5$	0.95	0.90
$i = 6$	0.93	0.93
$i = 7$	0.94	0.94
$i = 8$	0.94	0.98
$i = 9$	0.94	0.93
$i = 10$	0.96	0.93

Table 5.14: Fraction of times that true value of $B_{1,ij}$ fell into the 95% CI

$B_{1,ij}$	$j = 1$	$j = 2$
$i = 1$	0.93	0.95
$i = 2$	0.97	0.94
$i = 3$	0.94	0.97
$i = 4$	0.93	0.94
$i = 5$	0.89	0.97
$i = 6$	0.96	0.95
$i = 7$	0.94	0.95
$i = 8$	0.94	0.97
$i = 9$	0.92	0.98
$i = 10$	0.94	0.97

Table 5.15: Fraction of times that true value of $B_{1,ij}$ fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$
$i=1$	0.91	0.86	0.92	0.93	0.91	0.89	0.9	0.91	0.94	0.93
$i=2$	0.86	0.93	0.89	0.88	0.88	0.89	0.92	0.93	0.91	0.9
$i=3$	0.92	0.89	0.96	0.88	0.93	0.95	0.91	0.94	0.95	0.92
$i=4$	0.93	0.88	0.88	0.94	0.91	0.89	0.92	0.91	0.94	0.96
$i=5$	0.91	0.88	0.93	0.91	0.89	0.88	0.93	0.91	0.92	0.98
$i=6$	0.89	0.89	0.95	0.89	0.88	0.92	0.93	0.9	0.92	0.92
$i=7$	0.9	0.92	0.91	0.92	0.93	0.93	0.93	0.88	0.93	0.92
$i=8$	0.91	0.93	0.94	0.91	0.91	0.9	0.88	0.94	0.94	0.94
$i=9$	0.94	0.91	0.95	0.94	0.92	0.92	0.93	0.94	0.96	0.94
$i=10$	0.93	0.9	0.92	0.96	0.98	0.92	0.92	0.94	0.94	0.94

Table 5.16: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	94.28%	2.18
Σ	91.80%	3.22

Table 5.17: Total percentage of times that each parameter fell into the 95% CI

5.2.2 N = 50, K = 10

$B_{0,ij}$	$j=1$	$j=2$
$i=1$	0.92	-
$i=2$	0.95	0.97
$i=3$	0.95	0.95
$i=4$	0.96	0.94
$i=5$	0.93	0.94
$i=6$	0.97	0.95
$i=7$	0.97	0.94
$i=8$	0.94	0.99
$i=9$	0.95	0.94
$i=10$	0.93	0.96

Table 5.18: Fraction of times that true value of $B_{0,ij}$ fell into the 95% CI

$B_{1,ij}$	$j=1$	$j=2$
$i=1$	0.94	0.94
$i=2$	0.95	0.94
$i=3$	0.96	0.98
$i=4$	0.96	0.95
$i=5$	0.96	0.93
$i=6$	0.93	0.92
$i=7$	0.94	0.95
$i=8$	0.95	0.95
$i=9$	0.93	0.92
$i=10$	0.95	0.96

Table 5.19: Fraction of times that true value of $B_{1,ij}$ fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$
$i=1$	0.92	0.93	0.92	0.96	0.92	0.95	0.93	0.92	0.97	0.97
$i=2$	0.93	0.95	0.95	0.96	0.94	0.9	0.93	0.95	0.97	0.97
$i=3$	0.92	0.95	0.95	0.9	0.92	0.9	0.95	0.94	0.95	0.93
$i=4$	0.96	0.96	0.9	0.92	0.94	0.91	0.94	0.92	0.98	0.96
$i=5$	0.92	0.94	0.92	0.94	0.92	0.94	0.96	0.95	0.94	0.93
$i=6$	0.95	0.9	0.9	0.91	0.94	0.96	0.98	0.97	0.93	0.91
$i=7$	0.93	0.93	0.95	0.94	0.96	0.98	0.96	0.97	0.97	0.91
$i=8$	0.92	0.95	0.94	0.92	0.95	0.97	0.97	0.97	0.95	0.97
$i=9$	0.97	0.97	0.95	0.98	0.94	0.93	0.97	0.95	0.96	0.96
$i=10$	0.97	0.97	0.93	0.96	0.93	0.91	0.91	0.97	0.96	0.95

Table 5.20: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	94.76%	0.875
Σ	94.30%	0.733

Table 5.21: Total percentage of times that each parameter fell into the 95% CI

5.2.3 N = 20, K = 5

$B_{0,ij}$	$j=1$	$j=2$
$i=1$	0.95	-
$i=2$	0.94	0.85
$i=3$	0.89	0.96
$i=4$	0.95	0.88
$i=5$	0.92	0.87

Table 5.22: Fraction of times that true value of $B_{0,ij}$ fell into the 95% CI

$B_{1,ij}$	$j=1$	$j=2$
$i=1$	0.95	0.87
$i=2$	0.95	0.95
$i=3$	0.90	0.89
$i=4$	0.92	0.87
$i=5$	0.89	0.90

Table 5.23: Fraction of times that true value of $B_{1,ij}$ fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	0.99	0.94	0.97	0.94	0.95
$i=2$	0.94	0.88	0.95	0.90	0.92
$i=3$	0.97	0.95	0.94	0.93	0.95
$i=4$	0.94	0.90	0.93	0.94	0.95
$i=5$	0.95	0.92	0.95	0.95	0.97

Table 5.24: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	91.05%	2.91
Σ	94.08%	9.94

Table 5.25: Total percentage of times that each parameter fell into the 95% CI

5.2.4 N = 50, K = 5

$B_{0,ij}$	$j=1$	$j=2$
$i=1$	0.92	-
$i=2$	0.96	0.96
$i=3$	0.95	0.96
$i=4$	0.95	0.93
$i=5$	0.93	0.93

Table 5.26: Fraction of times that true value of $B_{0,ij}$ fell into the 95% CI

$B_{1,ij}$	$j=1$	$j=2$
$i=1$	0.92	0.95
$i=2$	0.94	0.94
$i=3$	0.95	0.96
$i=4$	0.93	0.96
$i=5$	0.90	0.95

Table 5.27: Fraction of times that true value of $B_{1,ij}$ fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	0.95	0.94	0.97	0.98	0.96
$i=2$	0.94	0.94	0.93	0.94	0.9
$i=3$	0.97	0.93	0.92	0.94	0.9
$i=4$	0.98	0.94	0.94	0.92	0.91
$i=5$	0.96	0.9	0.9	0.91	0.92

Table 5.28: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	94.16%	1.94
Σ	93.56%	5.72

Table 5.29: Total percentage of times that each parameter fell into the 95% CI

Configuration	Riemmanian distance ρ
N= 20, K = 10	0.073
N = 50, K = 10	0.047
N= 20, K = 5	0.16
N = 50, K = 5	0.054

Table 5.30: Comparing Riemmanian distances

5.3 p = 3 - Only intercept

5.3.1 N = 20, K = 10

$B_{2,ij}$	$j=1$	$j=2$	$j=3$
$i=1$	0.95	-	-
$i=2$	0.96	0.93	-
$i=3$	0.96	0.97	0.96
$i=4$	0.94	0.97	0.94
$i=5$	0.9	0.96	0.99
$i=6$	0.94	0.97	0.94
$i=7$	0.92	0.98	0.94
$i=8$	0.95	0.96	0.97
$i=9$	0.89	0.97	0.95
$i=10$	0.93	0.94	0.93

Table 5.31: Fraction of times that true value of B_{ij} fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$
$i=1$	0.97	0.95	0.96	0.92	0.94	0.94	0.96	0.95	0.96	0.97
$i=2$	0.95	0.96	0.93	0.94	0.94	0.92	0.98	0.96	0.97	0.92
$i=3$	0.96	0.93	0.92	0.95	0.94	0.95	0.95	0.97	0.94	0.95
$i=4$	0.92	0.94	0.95	0.95	0.95	0.92	0.92	0.96	0.98	0.96
$i=5$	0.94	0.94	0.94	0.95	0.95	0.96	0.93	0.94	0.98	0.95
$i=6$	0.94	0.92	0.95	0.92	0.96	0.96	0.96	0.94	0.97	0.98
$i=7$	0.96	0.98	0.95	0.92	0.93	0.96	0.96	0.97	0.91	0.94
$i=8$	0.95	0.96	0.97	0.96	0.94	0.94	0.97	0.93	0.97	0.95
$i=9$	0.96	0.97	0.94	0.98	0.98	0.97	0.91	0.97	0.95	0.97
$i=10$	0.97	0.92	0.95	0.96	0.95	0.98	0.94	0.95	0.97	0.94

Table 5.32: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	94.85%	1.90
Σ	95.03%	1.50

Table 5.33: Total percentage of times that each parameter fell into the 95% CI

5.3.2 N = 50, K = 10

$B_{0,ij}$	$j=1$	$j=2$	$j=3$
$i=1$	0.93	-	-
$i=2$	0.93	0.89	-
$i=3$	0.94	0.86	0.96
$i=4$	0.97	0.98	0.97
$i=5$	0.94	0.97	0.94
$i=6$	0.97	0.98	0.94
$i=7$	0.91	0.96	0.97
$i=8$	0.91	0.96	0.95
$i=9$	0.95	0.95	0.96
$i=10$	0.95	0.95	0.97

Table 5.34: Fraction of times that true value of B_{ij} fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$
$i=1$	0.93	0.96	0.97	0.98	0.98	0.95	0.94	0.95	0.96	0.97
$i=2$	0.96	0.93	0.97	0.98	0.97	0.96	0.94	0.95	0.95	0.98
$i=3$	0.97	0.97	0.94	0.95	0.96	0.94	0.88	0.97	0.93	0.97
$i=4$	0.98	0.98	0.95	0.98	0.96	0.97	0.94	0.93	0.96	0.98
$i=5$	0.98	0.97	0.96	0.96	0.94	0.96	0.94	0.96	0.98	0.96
$i=6$	0.95	0.96	0.94	0.97	0.96	0.98	0.95	0.95	0.94	0.96
$i=7$	0.94	0.94	0.88	0.94	0.94	0.95	1.0	0.97	0.93	0.95
$i=8$	0.95	0.95	0.97	0.93	0.96	0.95	0.97	0.93	0.92	0.96
$i=9$	0.96	0.95	0.93	0.96	0.98	0.94	0.93	0.92	0.95	0.96
$i=10$	0.97	0.98	0.97	0.98	0.96	0.96	0.95	0.96	0.96	0.95

Table 5.35: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	94.67%	1.07
Σ	95.51%	0.56

Table 5.36: Total percentage of times that each parameter fell into the 95% CI

5.3.3 N=20, K = 5

$B_{0,ij}$	$j=1$	$j=2$	$j=3$
$i=1$	0.92	-	-
$i=2$	0.89	0.96	-
$i=3$	0.96	0.97	0.98
$i=4$	0.96	0.96	0.92
$i=5$	0.95	0.97	0.88

Table 5.37: Fraction of times that true value of $B_{0,ij}$ fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	0.93	0.94	0.86	0.91	0.91
$i=2$	0.94	0.97	0.95	0.92	0.91
$i=3$	0.86	0.95	0.97	0.93	0.89
$i=4$	0.91	0.92	0.93	0.94	0.94
$i=5$	0.91	0.91	0.89	0.94	0.94

Table 5.38: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	95.14%	0.95
Σ	92.28%	0.83

Table 5.39: Total percentage of times that each parameter fell into the 95% CI

5.3.4 N=20, K = 5

$B_{1,ij}$	$j=1$	$j=2$	$j=4$
$i=1$	0.91	-	-
$i=2$	0.94	0.92	-
$i=3$	0.92	0.94	0.99
$i=4$	0.97	0.90	0.90
$i=5$	0.91	0.93	0.91

Table 5.40: Fraction of times that true value of $B_{0,ij}$ fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	0.94	0.96	0.9	0.93	0.92
$i=2$	0.96	0.95	0.95	0.95	0.93
$i=3$	0.9	0.95	0.94	0.93	0.93
$i=4$	0.93	0.95	0.93	0.91	0.89
$i=5$	0.92	0.93	0.93	0.89	0.9

Table 5.41: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	92.83%	0.69
Σ	92.88%	0.58

Table 5.42: Total percentage of times that each parameter fell into the 95% CI

Configuration	Riemmanian distance ρ
N= 20, K = 10	0.052
N = 50, K = 10	0.028
N= 20, K = 5	0.034
N = 50, K = 5	0.026

Table 5.43: Comparing Riemmanian distances - $p = 3$

5.4 p = 3 - normal covariate

5.4.1 N = 20, K = 10

$B_{0,ij}$	$j=1$	$j=2$	$j=3$
$i=1$	0.96	-	-
$i=2$	0.98	0.94	-
$i=3$	0.95	0.97	0.94
$i=4$	0.96	0.98	0.97
$i=5$	0.93	0.95	0.98
$i=6$	0.97	1.0	0.97
$i=7$	0.95	0.96	0.96
$i=8$	0.95	0.98	0.91
$i=9$	0.95	0.93	0.95
$i=10$	0.96	0.96	0.98

Table 5.44: Fraction of times that true value of $B_{0,ij}$ fell into the 95% CI

$B_{1,ij}$	$j=1$	$j=2$	$j=3$
$i=1$	0.95	0.97	0.96
$i=2$	0.95	0.99	0.92
$i=3$	0.94	0.96	0.97
$i=4$	0.94	0.92	0.94
$i=5$	0.95	0.97	0.94
$i=6$	0.95	0.99	0.96
$i=7$	0.96	0.97	0.96
$i=8$	0.96	0.99	0.98
$i=9$	0.98	0.94	1.0
$i=10$	0.92	0.98	1.0

Table 5.45: Fraction of times that true value of $B_{1,ij}$ fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$
$i=1$	0.98	0.89	0.93	0.93	0.96	0.95	0.92	0.93	0.92	0.93
$i=2$	0.89	0.96	0.91	0.89	0.92	0.95	0.91	0.91	0.91	0.92
$i=3$	0.93	0.91	0.96	0.94	0.96	0.97	0.92	0.97	0.93	0.98
$i=4$	0.93	0.89	0.94	0.97	0.91	0.92	0.92	0.95	0.9	0.98
$i=5$	0.96	0.92	0.96	0.91	0.92	0.95	0.93	0.94	0.93	0.95
$i=6$	0.95	0.95	0.97	0.92	0.95	0.97	0.93	0.96	0.91	0.96
$i=7$	0.92	0.91	0.92	0.92	0.93	0.93	0.94	0.95	0.88	0.94
$i=8$	0.93	0.91	0.97	0.95	0.94	0.96	0.95	0.96	0.9	0.97
$i=9$	0.92	0.91	0.93	0.9	0.93	0.91	0.88	0.9	0.93	0.9
$i=10$	0.93	0.92	0.98	0.98	0.95	0.96	0.94	0.97	0.9	0.96

Table 5.46: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	92.83%	1.87
Σ	93.41%	1.46

Table 5.47: Total percentage of times that each parameter fell into the 95% CI

5.4.2 N = 50, K = 10

$B_{0,ij}$	$j=1$	$j=2$	$j=3$
$i=1$	0.97	-	-
$i=2$	0.92	0.92	-
$i=3$	0.92	0.92	1.0
$i=4$	0.93	0.91	0.89
$i=5$	0.93	0.90	0.86
$i=6$	0.96	0.94	0.99
$i=7$	0.97	0.91	0.87
$i=8$	0.88	0.90	0.97
$i=9$	0.90	0.96	0.95
$i=10$	0.92	0.95	0.98

Table 5.48: Fraction of times that true value of $B_{0,ij}$ fell into the 95% CI

$B_{1,ij}$	$j=1$	$j=2$	$j=3$
$i=1$	0.89	0.98	0.99
$i=2$	0.94	0.95	0.99
$i=3$	0.92	0.96	0.80
$i=4$	0.90	0.92	0.94
$i=5$	0.90	0.93	0.99
$i=6$	0.90	0.96	0.94
$i=7$	0.91	0.94	0.99
$i=8$	0.93	0.93	0.91
$i=9$	0.97	0.96	0.99
$i=10$	0.95	0.93	1.0

Table 5.49: Fraction of times that true value of $B_{1,ij}$ fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$
$i=1$	0.95	0.95	0.98	0.96	0.94	0.94	0.95	0.94	0.91	0.92
$i=2$	0.95	0.94	0.95	0.97	0.93	0.96	0.95	0.94	0.94	0.94
$i=3$	0.98	0.95	0.95	0.98	0.94	0.95	0.98	0.94	0.92	0.95
$i=4$	0.96	0.97	0.98	0.94	0.96	0.97	0.94	0.95	0.98	0.96
$i=5$	0.94	0.93	0.94	0.96	0.95	0.93	0.95	0.94	0.9	0.94
$i=6$	0.94	0.96	0.95	0.97	0.93	0.96	0.94	0.95	0.96	0.97
$i=7$	0.95	0.95	0.98	0.94	0.95	0.94	0.93	0.96	0.91	0.93
$i=8$	0.94	0.94	0.94	0.95	0.94	0.95	0.96	0.95	0.92	0.95
$i=9$	0.91	0.94	0.92	0.98	0.9	0.96	0.91	0.92	0.96	0.96
$i=10$	0.92	0.94	0.95	0.96	0.94	0.97	0.93	0.95	0.96	0.96

Table 5.50: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	93.39%	3.33
Σ	94.67%	4.91

Table 5.51: Total percentage of times that each parameter fell into the 95% CI

5.4.3 N = 20, K = 5

$B_{0,ij}$	$j=1$	$j=2$	$j=4$
$i=1$	0.96	-	-
$i=2$	0.90	0.91	-
$i=3$	0.91	0.94	0.95
$i=4$	0.93	0.95	0.95
$i=5$	0.94	0.97	0.96

Table 5.52: Fraction of times that true value of $B_{0,ij}$ fell into the 95% CI

$B_{1,ij}$	$j=1$	$j=2$	$j=4$
$i=1$	0.96	0.93	0.92
$i=2$	0.90	0.95	0.96
$i=3$	0.96	0.94	0.93
$i=4$	0.91	0.95	0.91
$i=5$	0.93	0.94	0.96

Table 5.53: Fraction of times that true value of $B_{1,ij}$ fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	0.95	0.90	0.94	0.95	0.95
$i=2$	0.90	0.97	0.96	0.97	0.95
$i=3$	0.94	0.96	0.94	0.96	0.93
$i=4$	0.95	0.97	0.96	0.92	0.97
$i=5$	0.95	0.95	0.93	0.97	0.99

Table 5.54: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	93.78%	1.87
Σ	94.92%	1.91

Table 5.55: Total percentage of times that each parameter fell into the 95% CI

5.4.4 N = 50, K = 5

$B_{0,ij}$	$j=1$	$j=2$	$j=4$
$i=1$	0.90	-	-
$i=2$	0.94	0.93	-
$i=3$	0.88	0.95	0.98
$i=4$	0.92	0.96	0.98
$i=5$	0.93	0.95	0.93

Table 5.56: Fraction of times that true value of $B_{0,ij}$ fell into the 95% CI

$B_{1,ij}$	$j=1$	$j=2$	$j=4$
$i=1$	0.94	0.96	0.95
$i=2$	0.97	0.94	0.91
$i=3$	0.96	0.94	0.94
$i=4$	0.91	0.95	0.92
$i=5$	0.95	0.95	0.95

Table 5.57: Fraction of times that true value of $B_{1,ij}$ fell into the 95% CI

Σ_{ij}	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=1$	0.95	0.95	0.91	0.91	0.95
$i=2$	0.95	0.97	0.95	0.97	0.97
$i=3$	0.91	0.95	0.92	0.92	0.95
$i=4$	0.91	0.97	0.92	0.94	0.95
$i=5$	0.95	0.97	0.95	0.95	0.94

Table 5.58: Fraction of times that true value of Σ_{ij} fell into the 95% CI

Parameter	Global percentage	mean CI length
B	94.03%	1.22
Σ	94.32%	1.53

Table 5.59: Total percentage of times that each parameter fell into the 95% CI

Configuration	Riemmanian distance ρ
N= 20, K = 10	0.077
N = 50, K = 10	0.12
N= 20, K = 5	0.077
N = 50, K = 5	0.098

Table 5.60: Comparing Riemmanian distances - $p = 3$

As the tables suggest, the sampler is definitely able to retrieve the correct estimates of parameters.
However, it must be noted that the global level of approximate credible intervals tends to be lower than the nominal one, meaning that further investigation might be needed, for instance by generating more datasets and/ or increasing the number of iterations of the sampler.
Secondly, notice that the global percentage might be misleading if not properly interpreted: taking a global percentage when computing the approximate credible intervals result in a "pooling" effect. As a matter of fact, there are some parameters whose approximated CI percentage is even below 90%, meaning that, the sampler struggles more in finding the correct estimates.
As a final consideration, it must be noted that there seems to be no explicit relations between the less precisely estimated parameters and the geometric properties of the studied configurations, meaning again that further investigation might be needed to assess more precise results.

5.5 A real dataset - rats

We finally propose a real case application: the **rats** dataset from the package *shapes* of R.
The dataset is available at https://cran.r-project.org/web/packages/shapes/index.html and consists of 18 rats' skulls data from X rays, observed at 8 different times, for a total of $N = 144$ observation. Each observation consists of 8 2-dimensional landmarks.
The time instants are measured in days and are

$$t_i \in \{7, 14, 21, 30, 40, 60, 90, 150\}$$

As observed in [4], due to uneven spacing in time instants it might be useful to take the logarithm of t_i.
We study the regressive relation between the size-and-shape configuration $\boldsymbol{Y}_i$ and

$$z_i = \ln(t_i) \quad i = 1, \dots, N$$

In order to perform a stable analysis, we first apply the helmert sub-matrix of order $K = 8$ and then divide each entry of each observation by the standard deviation of the overall dataset's entries. Finally, SVD is performed and $\boldsymbol{Y}_i$ is retrieved for each observation.
We run the MCMC algorithm for 30000 iterations, discarding the first 10000 as burn-in and applying a thin of 10.
This procedures generates 2000 samples that are used to make inference.
Graphically, the result is shown in the next figure

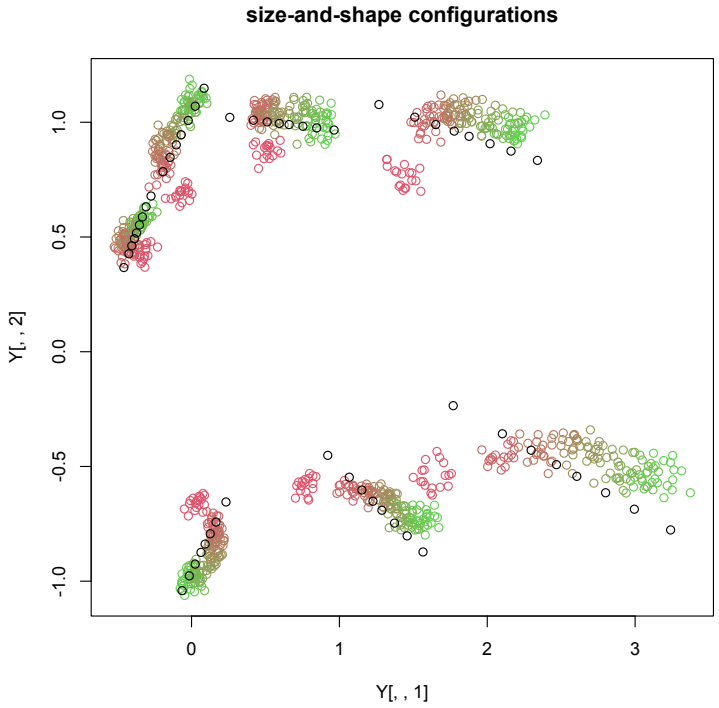


Figure 5.1: rats dataset: the real shape is given by the colored points, with time ranging from 7 days (red) to 150 days (green). Black dots are the linear approximation

The 95% credible intervals for the $\boldsymbol{B}$ parameters are here reported

$B_{0,ij}$	$j=1$	$j=2$
$i=1$	[0.623,0.647]	-
$i=2$	[0.779,0.810]	[-0.506,-0.469]
$i=3$	[0.394,0.443]	[-1.119,-1.087]
$i=4$	[-0.382, -0.296]	[-2.06, -2.020]
$i=5$	[-1.987,-1.906]	[-1.761,-1.687]
$i=6$	[-1.318, -1.277]	[-0.628,-0.578]
$i=7$	[-0.749, -0.719]	[0.438, 0.470]

Table 5.61: 95% CI for $B_{0,ij}$

$B_{1,ij}$	$j=1$	$j=2$
$i=1$	[0.025,0.049]	[-0.107,-0.064]
$i=2$	[0.030,0.066]	[-0.202,-0.147]
$i=3$	[-0.179,-0.109]	[-0.183, -0.143]
$i=4$	[-0.320, -0.195]	[-0.240, -0.198]
$i=5$	[-0.464,-0.347]	[-0.322,-0.202]
$i=6$	[-0.248, -0.197]	[-0.121,-0.040]
$i=7$	[-0.057, -0.022]	[0.120, 0.169]

Table 5.62: 95% CI for $B_{1,ij}$

It is worth noting that both positive and negative values of the regressive coefficients for the time covariate are present.
This means that landmarks' coordinates tend to move in more than a single direction, resulting in a shape whose landmarks not only become more far apart, but also change their relative positions.
Finally, observe that none of those intervals include the 0 element, meaning that, with a credibility of 95%, every landmark is affected by time variation, none of them excluded.

Chapter 6

Conclusions

In this document we showed how a size-and-shape model can be derived starting from simple geometrical considerations and how MCMC algorithms perfectly fit the Bayesian framework that is proposed.
We further explored the possibility of implementing such algorithms by means of Gibbs sampling, instead of the already proposed Metropolis approach.
The results shown in the previous section reveals that the sampler is able to capture the size-and-shape information, retrieving the correct parameters in an acceptable percentage of trials. Results also show that, even with few observations, some statistical significant relationships can be exploited.
This is one of the many advantages of using a specifically designed model, instead of a generic model-free approach. Moreover, the number of parameters is exiguous if compared to standard CNN's used in modern machine learning algorithms in image processing.
Conversely, this kind of approach presents some drawbacks: first of all, the model presented during this work is nothing but a standard linear regression, which means that a generalization might be needed to tackle more complex problems (such as binomial or Poisson models, that naturally fit into the GLM framework).
Secondly, it must be remarked that from computational point of view this models work well on standardized dataset but struggles on non-standardized datasets. This is due to the fact that rotations are quite sensible to small changes, when dealing with great distances: a small variation of an angle might produce a big arc length difference if the radius is big enough. As a consequence, the MCMC sampler produces highly correlated chains, which result in a great number of iterations to produce a small number of samples.
Last, using Euler angles parametrizations and Gram-Schmidt-like procedures for identification is totally fine when $p = 2$ or $p = 3$, but might become totally inappropriate when p grows due to the instability of these methods.

For these reasons, other parametrizations might seeked and explored, as well as different techniques to handle orthogonal matrices (such as Householder reflections or Givens Matrices when dealing with sparse systems).
Even with some limitations, these models might be of great importance when dealing with specific areas of study, such as biology, where model accountability might be an issue of great importance: the possibility of explicitly interpreting the parameters allow an expert to justify and clarify any choice that might be taken as a consequence of the model's results. This is another keypoint when dealing with the model-based approach: the relation between covariates and parameters is easily retrieved and can be explained to experts of those areas of study.
For these reasons, this kind of approach might be further investigated by proposing both a new theoretical framework (such as the already mentioned GLM) and a new implementation, allowing for instance the matrix R to be a general orthogonal matrix, insetad of a special one, letting the MCMC sampler to freely decide which kind of matrix to use.
In this way, not only the latent variable approach will be further exploited to ensure only size-and-shape inference, but also new types of models (such as the reflection shape one) might be investigated, allowing for a further study of specific phenomena.

Appendix A

Code implementation

A.1 The Julia environment

The MCMC algorithm was implemented in Julia 1.8.5. Julia is a compiled language that allows the user to take advantage of both a quick and easy implementation of the code and the code optimization due to compiler.
It is a valid alternative to many known languages, such as Python, as it allows the user to build custom types that will run just as fast as the built in ones.
Moreover, de-vectorized code will run as fast as the vectorized one. There are however some disadvantages, such as the absence of a real object oriented framework, that might make the coding a bit less straightforward.
The code was implemented trying to follow the main guidelines given from Julia developers, that are essentially well-known precautions of programming: declare types of the variables, avoid different types interaction as they require every time a conversion ad make use of functions to execute the more expensive tasks.
Having all these information in mind, here we present and comment the code in every of its function.

A.2 Main functions

The mcmc file contains each and every function that was used during the applications.
In the following, we will very briefly present the main functions that allow the sampler tho work.

A.2.1 mcmc setup

Input:

- **I_max::Int64** max number of iterations
- **burn_in::Int64** burn-in size
- **thin::Int64** thin size
- **d::Int64** number of covariates
- **K::Int64** number of landmarks
- **p::Int64** number of coordinates
- **N::Int64** number of observations
- **M_prior::Array{Float64}** Array containing prior mean for B
- **V_prior::Array{Float64}** Array containing prior variance for B

Output:

- **B::Array{Float64}**
- **M::Array{Float64}**
- **V::Array{Float64}**
- **Sigma_est::Array{Float64}**
- **theta::Array{Float64}**
- **R::Array{Float64}**
- **X::Array{Float64}**

```
1 function mcmc_setup(I_max::Int64,burn_in::Int64,thin::Int64,d::Int64,K::
      Int64,p::Int64,N::Int64,M_prior::Array{Float64},V_prior::Array{
      Float64})
2       #----Parameters initialization-----#
3       # - Regressive coefficients - #
4       n_samples = size([i for i = burn_in+1:thin:I_max])[1]
5       B = zeros(n_samples,d,K,p);
6       B[1,1,:,:] = ones(K,p)
7
```

```
8       #Priors for B
9       #M = zeros(p,K*d);
10      M = M_prior
11      V = zeros(p,K*d,K*d);
12      for l = 1:p
13          V[l,:,:] = V_prior;
14      end
15
16      #Variance matrix
17      Sigma_est = zeros(n_samples,K,K)
18      Sigma_est[1,:,:] = I(K)
19
20
21      # - Angles and rotations - #
22      R = zeros(n_samples,N,p,p);
23      #Angles
24      theta = zeros(n_samples,N,3)
25      #Initialization
26      theta[1,:,1] = ones(N);
27      theta[1,:,2] = ones(N);
28      theta[1,:,3] = ones(N);
29
30      #Initialization of R
31      if p == 3
32          for s = 1:N
33              #R[1,s,:,:] = Rotation(theta[1,s,1], theta[1,s,2], theta[1,s,3]);
34              R[1,s,:,:] = Matrix(I(3))
35          end
36      elseif p == 2
37          for s = 1:N
38              #R[1,s,:,:] = [cos(theta) sin(theta); -sin(theta) cos(theta)];
39              R[1,s,:,:] = Matrix(I(2))
40          end
41      end
42
43      #Tensor containing sampled configurations
44      X = zeros(I_max,N,K,p)
45      X[1,:,:,:] = ones(N,K,p)
46      return B,M,V,Sigma_est,theta,R,X
47  end
```

A.2.2 mcmc

Function used to obtain samples from the posterior distributions.

Input

- **I_max::Int64** Maximum number of iterations of the sampler
- **burn_in::Int64** Number of samples to discard (before thin)
- **thin::Int64** Number of sample to discard between to consecutive saved samples
- **d::Int64** Number of covariates
- **K::Int64** Number of landmarks
- **p::Int64** Number of coordinates
- **N::Int64** Number of observations
- **z::Array{Float64}** Matrix of covariates
- **Z::Array{Float64}** Design matrix
- **Y::Array{Float64}** True shape configurations
- **nu::Int64** Prior parameter of $\boldsymbol{\Sigma}$
- **Psi::Array{Float64}** Prior parameter of $\boldsymbol{\Sigma}$
- **M_prior::Array{Float64}** Prior parameter of $\boldsymbol{B}$
- **V_prior::Array{Float64}** Prior parameter of $\boldsymbol{B}$
- **original::Int64 = 0** Flag: set = 1 to use original data instead of simulated
- **samples::Array{Float64}=zeros(N,K,p)** Array containing the samples from the dataset (used only if original = 1)
- **B_true::Union{Array{Float64},Nothing}** = nothing True regressive coefficients
- **Sigma_true::Union{Array{Float64},Nothing}** = nothing True Covariance matrix

- **theta_true::Union{Array,Nothing}** = nothing True vector of angles
- **theta_sim::Union{Array,Nothing}** = nothing Vector of flags: each entry set = 1 will induce the samplign of teh corresponding angle
- **beta_sim::Int64 = 0** Flag to simulate B
- **Sigma_sim::Int64 = 0** Flag to simulate $\mathbf{\Sigma}$

Output:

- **B::Array{Float64}** Array containing the regressive coefficients' samples
- **Sigma_est::Array{Float64}** Array containing the covariance matrix samples
- **theta::Array{Float64}** Array containing angles samples
- **R::Array{Float64}** Array containing rotation samples
- **X::Array{Float64}** Array containing estimated configuration during sampling

```
1  function mcmc(I_max::Int64, burn_in::Int64, thin::Int64, d::Int64,K::Int64,
       p::Int64,N::Int64,z::Array{Float64},Z::Array{Float64},Y::Array{
       Float64},nu::Int64,Psi::Array{Float64},M_prior::Array{Float64},
       V_prior::Array{Float64}, original::Int64 = 0,samples::Array{Float64
       }=zeros(N,K,p), B_true::Union{Array{Float64},Nothing} = nothing,
       Sigma_true::Union{Array{Float64},Nothing} = nothing, theta_true::
       Union{Array,Nothing} = nothing, theta_sim::Union{Array,Nothing}
       = nothing, beta_sim::Int64 = 0, Sigma_sim::Int64 = 0)
2  B,M,V,Sigma_est,theta,R,X = mcmc_setup(I_max,burn_in,thin,d,K,p,N,
       M_prior,V_prior);
3
4      X_last = X[1,:,:,:]
5      R_last = R[1,:,:,:]
6      B_last = B[1,:,:,:]
7      Sigma_last = Sigma_est[1,:,:]
8      theta_last = theta[1,:,:]
9      k = 1
10     for i = 2:I_max
11         if original == 0
12             X_last = sample_update!(X_last,R_last,Y)
```

```
13          elseif original == 1
14              X_last = samples
15
16          end
17
18          if beta_sim == 1
19              B_last=mcmc_B!(N,p,K,d,Sigma_last,Z,V,M,X_last);
20          else
21              B_last = B_true
22          end
23
24          if Sigma_sim == 1
25              Sigma_last=mcmc_Sigma!(N,K,p,nu_prior,Psi_prior,X_last,
                    Z,B_last);
26          else
27              Sigma_last= Sigma_true
28          end
29
30          if original == 0 && isnothing(theta_true)
31              theta_last, R_last = mcmc_theta!(N,B_last,z,Sigma_last,
                    theta_last);
32          elseif !isnothing(theta_true)
33              theta_last, R_last = mcmc_theta!(N,B_last,z,Sigma_last,
                    theta_last, theta_true, theta_sim)
34          end
35
36          if i%1000 == 0
37              print("Iteration counter: ",i, '\n')
38          end
39
40          if i > burn_in
41              if (i-burn_in)%thin == 0
42                  X[k,:,:,:] = X_last
43                  theta[k,:,:] = theta_last
44                  R[k,:,:,:] = R_last
45                  B[k,:,:,:] = B_last
46                  Sigma_est[k,:,:] = Sigma_last
47                  k +=1
48              end
49          end
50      end
51
52      return B, Sigma_est, theta, R, X
53  end
```

A.2.3 mcmc_theta

Function to obtain samples from the posterior distribution of θ_is.

Input

- **N::Int64** NUmber of configurations
- **B::Array{Float64}** Value of $\boldsymbol{B}$
- **z::Array{Float64}** vector of covariates
- **Sigma::Array{Float64}** Value of $\boldsymbol{\Sigma}$
- **theta_last::Array{Float64}** last sample of $\boldsymbol{\theta}$
- **theta_true::Union{Array{Float64},Nothing}=nothing** vector containing the true values of angles
- **theta_sim::Union{Array{Int64},Nothing}=nothing** vector of flags to choose which angles to simulate

Output:

- **theta::Array{Float64}** array containing the sampled angles
- **R::Array{Float64}** Array containing the corresponding rotation matrices

```
1  function mcmc_theta!(N::Int64,B::Array{Float64},z::Array{Float64},
       Sigma::Array{Float64},theta_last::Array{Float64}, theta_true::Union
       {Array{Float64},Nothing}=nothing, theta_sim::Union{Array{Int64},
       Nothing}=nothing)
2      d = size(B)[1]
3      K = size(B)[2]
4      p = size(B)[3]
5      theta = zeros(N,3)
6      R = zeros(N,p,p)
7      I_Sigma = inv(Sigma[:,:])
8      for s = 1:N
9
10         m = zeros(K,p)
11         for h = 1:d
12             m += z[s,h]*B[h,:,:]
13         end
14
```

```
15          A = m'*I_Sigma*Y[s,:,:]
16
17          theta1 = theta_last[s,1]
18
19          if p == 2
20              theta2 = 0
21              theta3 = 0
22          elseif p == 3
23              theta2 = theta_last[s,2]
24              theta3 = theta_last[s,3]
25          end
26
27          #Rotation matrices
28
29          if p == 3
30              R1 = [
31                  cos(theta1) -sin(theta1) 0;
32                  sin(theta1) cos(theta1) 0;
33                  0 0 1
34              ]
35
36              R2 = [
37              1 0 0;
38              0 cos(theta2) -sin(theta2);
39              0 sin(theta2) cos(theta2)
40
41              ]
42
43              R3 = [
44              cos(theta3) -sin(theta3) 0;
45              sin(theta3) cos(theta3) 0;
46              0 0 1
47              ]
48
49              H = A'*R3*R2
50
51              if isnothing(theta_true)
52                  #Sample theta_1
53                  theta[s,1] = sample(H,1)
54                  #Sample theta_2
55                  theta[s,2] = sample(D,2)
56                  #Sample theta_3
57                  theta[s,3] = sample(L,1)
58                  #Build R
59                  R[s,:,:] = Rotation(theta[s,1], theta[s,2], theta[s,3])
```

```
60              else
61                  if theta_sim[1] == 0
62                      #Check that angle lies in [0,2pi]
63                      if(theta_true[s,1]<0)
64                          theta_true[s,1] += 2pi
65                      end
66                      theta[s,1] = theta_true[s,1]
67                  else
68                      theta[s,1] = sample(H,1)
69                  end
70
71                  if theta_sim[2] == 0
72                      #Check that angle lies [0,pi]
73                      if(theta_true[s,2]<0)
74                          theta_true[s,2] += pi
75                      end
76                      theta[s,2] = theta_true[s,2]
77                  else
78                      R1 = RX(theta[s,1])
79                      D = R1*A'*R3
80                      theta[s,2] = sample(D,2)
81                  end
82
83                  if theta_sim[3] == 0
84                      #Check that angle lies in [0,2pi]
85                      if(theta_true[s,3]<0)
86                          theta_true[s,3] += 2pi
87                      end
88                      theta[s,3] = theta_true[s,3]
89                  else
90                      R2 = RZ(theta[s,2])
91                      L = R2*R1*A'
92                      theta[s,3] = sample(L,1)
93                  end
94                  R[s,:,:] = Rotation(theta[s,1], theta[s,2], theta[s,3])
95              end
96          end
97          if p == 2
98              if theta_sim[1] == 0
99                  #while(theta_true[s,1]<0)
100                  # theta_true[s,1] += 2pi
101                 #end
102                 theta[s,1] = theta_true[s,1]
103                 theta[s,2] = 0
104                 theta[s,3] = 0
```

```
105                 R[s,:,:] = [cos(theta[s,1]) -sin(theta[s,1]); sin(theta[s,1]) cos
                        (theta[s,1]) ]
106             else
107                 a = A[1,1]+A[2,2]
108                 b = A[2,1]-A[1,2]
109                 rho = sqrt(a^2+b^2)
110                 gamma = atan(b,a)
111                 while(gamma < 0)
112                     gamma += 2pi
113                 end
114                 #println(rho)
115                 #gamma = 2pi - gamma
116                 theta[s,1] = rand(VonMises(gamma,rho))
117                 while(theta[s,1]<0)
118                     theta[s,1] += 2*pi
119                 end
120                 R[s,:,:] = [cos(theta[s,1]) -sin(theta[s,1]); sin(theta[s,1]) cos
                        (theta[s,1]) ]
121             end
122         end
123     end
124     return theta, R
125 end
```

A.2.4 mcmc_B

Function to obtain samples of B, conditioned to everything else

Input

- **N::Int64** Number of configurations
- **p::Int64** Number of coordinates
- **K::Int64**
- **d::Int64** Number of landmarks
- **Sigma::Array{Float64}** Given value of $\mathbf{\Sigma}$
- **Z::Array{Float64}** Desing matrix
- **V::Array{Float64}** Prior covariance matrix for B
- **M::Array{Float64}** Prior mean vector for B

- **X::Array{Float64}** Array containing the simulated rotation-shapes configuration

Output:

- **B::Array{Float64}** Array containing samples of regressive coefficients

```
function mcmc_B!(N::Int64,p::Int64,K::Int64,d::Int64,Sigma::Array{
    Float64},Z::Array{Float64},V::Array{Float64},M::Array{Float64},X::
    Array{Float64})
    B = zeros(d,K,p)
    M_s = zeros(p,K*d,1);
    V_s = zeros(p,K*d,K*d);
    I_Sigma = inv(Sigma)
    for l = 1:p
        V_acc = zeros(K*d,K*d)
        M_acc = zeros(K*d,1)
        for s = 1:N
            V_acc += Z[s,:,:]'*I_Sigma*Z[s,:,:]
            M_acc += Z[s,:,:]'*I_Sigma*X[s,:,l]
            #V_s[l,:,:] = V_s[l,:,:] + Z[s,:,:]'*I_Sigma*Z[s,:,:]
            #M_s[l,:] = M_s[l,:] + Z[s,:,:]'*I_Sigma*X[s,:,l]
        end
        V_s[l,:,:] += V_acc
        M_s[l,:] += M_acc
        V_s[l,:,:] = V_s[l,:,:] +inv(V[l,:,:])
        M_s[l,:] = M_s[l,:] +inv(V[l,:,:])*M[l,:]

        V_s[l,:,:] = inv(V_s[l,:,:]);
        V_s[l,:,:] = Hermitian(V_s[l,:,:])
        M_s[l,:] = V_s[l,:,:]*M_s[l,:];

    #Campiono dalla full-conditional di beta_l
        B[:,:,l] = rand(MvNormal(
            M_s[l, :], V_s[l, :, :]
        ))
    end
    return B
end
```

A.2.5 mcmc_Sigma

Function to obtain samples from the posterior distribution of $\mathbf{\Sigma}$, given all the other parameters.

Input

- **N::Int64** Number of configurations
- **K::Int64** Number of landmarks
- **p::Int64** Number of coordinates
- **nu::Int64** Prior parameter for $\boldsymbol{\Sigma}$
- **Psi::Array{Float64}** Prior parameter for $\boldsymbol{\Sigma}$
- **X::Array{Float64}** Array containing the simulated rotation-shapes configurations
- **Z::Array{Float64}** Design matrix
- **B::Array{Float64}** Given value of $\boldsymbol{B}$

Output:

- **Sigma::Array{Float64}** Array containing samples of covariance matrix

```
1  function mcmc_Sigma!(N::Int64,K::Int64,p::Int64,nu::Int64,Psi::Array{
       Float64},X::Array{Float64},Z::Array{Float64},B::Array{Float64})
2          #Ricavo i parametri necessari per campionare dalla full
               conditional di Sigma
3          nu_s = nu+N*p;
4
5          Psi_s = zeros(K,K);
6          for s = 1:N
7              for l = 1:p
8                  Psi_s = Psi_s + (X[s,:,l]-Z[s,:,:]*vec(B[:,:,l]))*(X[s,:,l]-Z[s
                       ,:,:]*vec(B[:,:,l]))'
9              end
10         end
11         Psi_s = Psi_s + Psi
12
13         #Campiono dalla full di Sigma
14          Sigma = rand(
15             InverseWishart(nu_s, Psi_s)
16         )
17         return Sigma
18 end
```

A.2.6 sample_update

This function computes the actual estimate of YR^T for each configuration of landmarks in the dataset. The function is implemented by taking advantage of a vectorized form that allows for a significant time gain if compared to a standard for loop.

Input

- **X::Array{Float64}** Array containing estimates of X
- **R::Array{Float64}** Array containing samples of R
- **Y::Array{Float64}** Array containing samples of Y

Output:

- **X::Array{Float64}** Array containing updated configurations using the rotations given

```
function sample_update!(X::Array{Float64},R::Array{Float64},Y::Array{
    Float64})
    N,K,p = size(Y)
    Y_v = [Y[i,:,:] for i in 1:N]
    R_v = [R[i,:,:]' for i in 1:N]
    X = permutedims(reshape(reduce(hcat, Y_v.*R_v),K,p,N),(3,1,2))
    return X
end
```

A.2.7 make_dataset

Function that builds a synthetic dataset with the given parameters, while simultaneously computing the size-and-shape decomposition.

Input:

- **N::Int64** Number of configurations
- **d::Int64** Number of covariates
- **K::Int64** Number of landmarks
- **p::Int64** Number of coordinates

- **z::ArrayFloat64** Array of covariates
- **B::ArrayFloat64** Array of regressive coefficients
- **VarCov::ArrayFloat64** Covariance matrix

Output:

- **samples::Array{Float64}** Dataset's samples
- **Y::Array{Float64}** Array of size-and-shape configurations
- **R_true::Array{Float64}** Array of true rotations
- **theta_true::Array{Float64}** Array of true angles

```
1  function makedataset(N::Int64,d::Int64,K::Int64,p::Int64,z::Array{Float64},
       B::Array{Float64},VarCov::Array{Float64})
2      samples = zeros(N,K,p);
3      Y = zeros(N,K,p)
4      R_true = zeros(N,p,p)
5      #Tensore degli angoli
6      theta_true = zeros(N,3)
7      #Sample N elements from a multivariate normal distribution
8      for i = 1:N
9          mu = zeros(K,p)
10         for h = 1:d
11             mu += z[i,h]*B[h,:,:]
12         end
13         #display(mu[1,1])
14         samples[i,:,:] = reshape(
15             rand(
16             MvNormal(vec(mu), VarCov)
17             ),
18             K,p)
19             #display(samples[i,1,1])
20     end
21
22     #Remove rotation
23     for i = 1:N
24         global P = zeros(p,p)
25         F = svd(samples[i,:,:])
26         U = F.U
27         V = F.V
28         #Ensure that V lies in SO(p)
29         if(det(F.Vt) < 0)
```

```
30                      if p == 3
31                          V[:,2] = -V[:,2]
32                          U[:,2] = -U[:,2]
33                      elseif p == 2
34                          V[:,2] = -V[:,2]
35                          U[:,2] = -U[:,2]
36                      end
37                  elseif (det(F.Vt) > 0)
38                      P = I(p)
39                  elseif (det(F.Vt == 0))
40                      print("Matrix is singular!")
41                      break
42                  end
43              Y[i,:,:] = U*Diagonal(F.S)
44              R_true[i,:,:] = V
45              theta_true[i,:] = angles(R_true[i,:,:])
46              end
47          return samples, Y, R_true, theta_true
48      end
```

A.3 Identification functions

A.3.1 identify

Function that computes the identified version of the regressive coefficients.

Input :

- **B::Array{Float64}** Array containing the samples of regressive coefficients

```
1   function identify(B::Array{Float64})
2       dims = size(B)
3       I_max = dims[1]
4       d = dims[2]
5       K = dims[3]
6       p = dims[4]
7       B_identified = zeros(I_max,d,K,p)
8       for i = 1:I_max
9           #function that computes the matrix needed to identify
10          V = get_V(B[i,1,:,:])
11          for h=1:d
12              B_identified[i,h,:,:] = B[i,h,:,:]*V
```

```
13          end
14      end
15      return B_identified
16  end
```

A.3.2 identify_t

Function that, given an array of angles, ensures that the angles are within the range $[0, 2\pi]$

Input:

- **theta::Array{Float64}** vector of angles

Output

- **theta::Array{Float64}** vector of identified angles

```
1   function identify_t!(theta::Array{Float64})
2
3       while theta[1] <0
4           theta[1] += 2pi
5       end
6       while theta[2] <0
7           theta[2] += pi
8       end
9       while theta[3] <0
10          theta[3] += 2pi
11      end
12
13      while theta[1] > 2pi
14          theta[1] %= 2pi
15      end
16      while theta[2] > pi
17          theta[2] %= pi
18      end
19      while theta[3] > 2pi
20          theta[3] %= 2pi
21      end
22
23      return theta
24  end
```

A.3.3 identify_angles

Extension of the previous function to a generic array with multiple dimensions

Input:

- **theta::Array{Float64}** Array of angles

Output

- **theta::Array{Float64}** Array of identified angles

```
function identify_angles(theta::Array{Float64})
    N = size(theta)[1]
    for i = 1:N
        theta[i,:] = identify_t!(theta[i,:])
    end
    return theta
end
```

Function that computes the matrix needed to perform identification of both the regressive coefficients and the rotations matrices. It is essentially a Gram-Schmidt orthogonalization.

A.3.4 get_V

Input:

- **B::Array{Float64}** Array containing samples of regressive coefficients

output

- **V::Array{Float64}** Matrix used to perform identification

```
function get_V(B::Array{Float64})
    p = size(B)[2]
    A = B'
    V = zeros(p,p);
    V[:,1] = A[:,1]/norm(A[:,1])
    V[:,2] = A[:,2] - (A[:,2]'*V[:,1])*V[:,1]
    V[:,2] = V[:,2]/norm(V[:,2])
    if p == 3
```

```
9           V[:,3] = A[:,3] - ((A[:,3]'*V[:,1])*V[:,1]) - ((A[:,3]'*V[:,2])*V[:,2])
10          V[:,3] = V[:,3]/norm(V[:,3])
11          #Last column is chosen to ensure that the matrix lies in SO(3)
12          if (det(V) < 0)
13              V[:,3] = -V[:,3]
14          end
15      elseif p == 2
16          if(det(V) < 0)
17              V[:,2] = -V[:,2]
18          end
19      end
20      return V
21  end
```

A.3.5 identify_R_angles

Function used to identify rotations paragraphInput:

- **B::Array{Float64}** Array containing the samples of regressive coefficients
- **R::Array{Float64}** Array containing the samples of R

Output

- **R_new::Array{Float64}** Array of identified rotations
- **R_new::Array{Float64}** Array of corresponding identified angles

```
1   function identify_R_angles(B::Array{Float64},R::Array{Float64})
2       R_new = copy(R)
3       dim = size(R)
4       N = dim[1]
5       S = dim[2]
6       K = size(B)[3]
7       p = dim[3]
8       thetas = zeros((N,S,3))
9       for i = 1:N
10          G = get_V(reshape(B[i,1,:,:],K,p))
11          for s = 1:S
12              R_new[i,s,:,:] = G'*R[i,s,:,:]
13              #R_new[i,s,:,:] = R[i,s,:,:]*G
14              thetas[i,s,:] = identify_t!(angles(copy(R_new[i,s,:,:])))
15          end
```

```
16      end
17      return R_new, thetas
18  end
19
20  function identify_R_angles_true(B_true::Array{Float64},R_true::Array{
        Float64})
21      dim = size(R_true)
22      S = dim[1]
23      K = dim[2]
24      p = dim[3]
25
26      R_true_new = copy(R_true)
27      thetas = zeros((S,3))
28      G = get_V(B_true[1,1,:,:])
29      for s = 1:S
30          R_true_new[s,:,:] = G'*R_true[s,:,:]
31          thetas[s,:] = identify_t!(angles(R_true_new[s,:,:]))
32      end
33      return R_true_new, thetas
34  end
```

A.3.6 identify_param

Function that identifies all the parameters of the Markov Chain (regressive coefficients, angles, rotations)

Input:

- **Y::Array{Float64}**
- **B::Array{Float64}**
- **B_true::Array{Float64}**
- **Sigma::Array{Float64}**
- **Sigma_true::Array{Float64}**
- **R::Array{Float64}**
- **R_true::Array{Float64}**

```
function identify_params(Y::Array{Float64},B::Array{Float64}, B_true::
    Array{Float64}, Sigma::Array{Float64},Sigma_true::Array{Float64},
    R::Array{Float64},R_true::Array{Float64})

    I_max,K,p = size(B)[[1,3,4]]
    N = size(Y)[1]
    B_id = identify(B)
    B_true_id = identify(B_true)
    R_id, theta_id = identify_R_angles(B,R)
    R_true_id,theta_true_id = identify_R_angles_true(B_true,R_true)

    X_id = zeros(I_max,N,K,p)
    for i = 2:I_max
        X_id[i,:,:,:] = sample_update!(X_id[i,:,:,:],R_id[i-1,:,:,:],Y)
        #=for s =1:N
            X_id[i,s,:,:] = X_id[i,s,:,:]
        end
        =#
    end


    samples_id = zeros(N,K,p)
    samples_id = sample_update!(samples_id,R_true_id,Y)
    #=for s = 1:N
        samples_id[s,:,:] = samples_id[s,:,:]
    end
    =#
    return samples_id, X_id, B_id, B_true_id, R_id, R_true_id,
        theta_id, theta_true_id
end
```

A.4 Misc functions

A.4.1 Helm

Input:

- **k::Int64** order of Helmert sub-matrix

Output

- **H::Array{Float64}** Helmert sub-matrix of order k

```
function Helm(k)
    H = zeros(k-1,k);
    for j = 1:k-1
        dj = 1/sqrt(j*(j+1))
        H[j,1:j] = -dj*ones(1,j);
        H[j,j+1] = j*dj
    end
    return H
end
```

A.4.2 Rotation

Function that returns the 3D rotation matrix associated to three Euler angles (ZXZ convention).

Input:

- **theta1::Float64** First Euler angle
- **theta2::Float64** Second Euler angle
- **theta3::FLoat64** Third Euler angle

Output:

- **R::Array{Float64}** Rotation matrix corresponding to the three Euler angles

```
function Rotation(theta1,theta2,theta3)
    #Funzione che prende in input i 3 angoli di Eulero con convenzione
        ZXZ e restituisce la corrispondente matrice di rotazione in R3
    R1 = [
        cos(theta1) -sin(theta1) 0;
        sin(theta1) cos(theta1) 0;
        0 0 1
    ]

    R2 = [
     1 0 0;
     0 cos(theta2) -sin(theta2);
     0 sin(theta2) cos(theta2)

    ]

```

```
16      R3 = [
17          cos(theta3) -sin(theta3) 0;
18          sin(theta3) cos(theta3) 0;
19          0 0 1
20      ]
21
22      return R3*R2*R1
23  end
```

A.4.3 RZ and RX

Functions that computes, respectively, the rotation matrix on the Z and X axis

Input:

- **theta::Float64**

Output:

- **R::Array{Float64}**: Rotation matrix corresponding to the given angle

```
1   function RZ(theta1::Float64)
2       R = [
3           cos(theta1) -sin(theta1) 0;
4           sin(theta1) cos(theta1) 0;
5           0 0 1
6       ]
7       return R
8   end
9
10  function RX(theta2::Float64)
11      R2 = [
12          1 0 0;
13          0 cos(theta2) -sin(theta2);
14          0 sin(theta2) cos(theta2)
15
16          ]
17      return R
18  end
```

A.4.4 angles

Function that decomposes a generic rotation matrix into its Euler angles

Input:

- **R::Array{Float64}** Rotation matrix to decompose

Output:

- **[theta1,theta2,theta3]::Array{Float64}**: vector containing the three Euler angles. If $p = 2$, the last 2 angles are automatically set to zero.

```
function angles(R::Array{Float64})
        p = size(R)[2]
        if p == 3
            theta2 = acos(R[3,3])
            if theta2 == 0
                theta1 = 0
                theta3 = 2pi
            else
                #theta3 = atan(R[1,3]/sin(theta2),R[2,3]/sin(theta2))
                #theta1 = atan(R[3,1]/sin(theta2), -R[3,2]/sin(theta2))
                theta3 = atan(R[1,3]/sin(theta2),-R[2,3]/sin(theta2))
                theta1 = atan(R[3,1]/sin(theta2), R[3,2]/sin(theta2))
            end
        elseif p == 2
            theta1 = atan(R[2,1],R[1,1])
            theta2 = 0
            theta3 = 0
        end

    return [theta1,theta2,theta3]
end
```

A.4.5 sample

Function used to sample angles from the corresponding distribution.

Input:

- **B::Array{Float64}** Matrix of weights that builds the distributions
- **type::Int** Flag: if 1, the function samples from a Von-Mises, if 2 the function samples from a Von-Mises-like distribution, using accept-reject sampling.

Output:

- **y::Float64** sample from the desired distribution

```
function sample(B::Array{Float64},type::Int)
    flag = 0
    i = 0

    if type == 1
        a = B[1,1]+B[2,2]
        b = B[1,2]-B[2,1]
    elseif type == 2
        a = B[2,2]+B[3,3]
        b = B[2,3]-B[3,2]
    else
        return NaN
    end

    #Compute rho, sin(gamma) e cos(gamma)
    rho = sqrt(a^2 + b^2)
    cgamma = a/rho
    sgamma = b/rho


    #Using arctan2 to ensure that angle is in the right domain, when
        sampling from a Von-Mises
    if type == 1
        gamma = atan(sgamma,cgamma)
        if(gamma <0)
            gamma += 2pi
        end
    else
        #If sampling from a simil-Von mises is [o ,), which is fine with
            arccos
        gamma = acos(cgamma)
    end


    while flag == 0
        #If rho = 0 the the distribution is degenerate and uniform in [o,
            2]
        if rho == 0
            y = rand(Uniform(0,2pi))
        else
            #Sampling from a VonMises, caring that Julia samples from
                [-, ]
```

```
39              y = rand(VonMises(gamma,rho))
40          end
41          #Sampling a uniform on (0,1)
42          u = rand(Uniform(0,1))
43
44          if type == 2
45              #Ensure that sample lies in [o ,)
46              if (y<0)
47                  y+=pi
48              end
49              y = y%pi
50          end
51
52
53          #If type = 2 use VonMises as a Kernel and do accept-reject,
                where the ratio is equak to sin(y)
54          if( (u <= sin(y)) && (type == 2))
55              return y
56          #Se type = 1 simply use the VonMises sample, ensuring that it
                lies in [o, 2]
57          elseif type == 1
58              if(y < 0)
59                  y += 2*pi
60              end
61              return y
62          i = i+1
63          end
64
65      end
66  end
```

A.4.6 a_mises

Input:

- **rho::Float64**: concentration parameter
- **gamma::Float64**: mean of the distribution
- **x::Float64**: point in which we want to compute the density

Output:

- **f::Float64** value of pdf in x

```
function a_mises(rho::Float64,gamma::Float64,x::Float64)
    return exp(rho*cos(x-gamma))/(2pi*besseli(0,x))*sin(x)
end
```

A.4.7 decomp_dataset

Function that decomposes each observation of a given dataset into size-and-shape and rotation

Input:

- **samples::ArrayFloat64** original dataset
- **N::Int64** Number of observations
- **d::Int64** Number of covariates
- **K::Int64** Number of landmarks
- **p::Int64** Number of coordinates

Output:

- **Y::Array{Float64}** size-and-shape arrays
- **R_true::Array{Float64}** Array containing the true rotations matrix
- **theta_true::Array{Float64}** Array containing the true angles

```
function decomp_dataset(samples::Array{Float64},N::Int64,d::Int64,K::
    Int64,p::Int64)
    #size-and-shape tensor
    Y = zeros(N,K,p)
    #True rotations tensor
    R_true = zeros(N,p,p)
    #Angles tensor
    theta_true = zeros(N,3)

    #Removing rotation
    for i in 1:N
        global P = zeros(p,p)
        F = svd(samples[i,:,:])
        U = F.U
```

```
14          V = F.V
15          #If V is not in SO(3) change the sign of one column of V and the
                respective column of U
16          if(det(F.Vt) < 0)
17              V[:,2] = -V[:,2]
18              U[:,2] = -U[:,2]
19          end
20          Y[i,:,:] = U*Diagonal(F.S)
21          R_true[i,:,:] = V
22          theta_true[i,:] = angles(R_true[i,:,:])
23          end
24      return Y, R_true, theta_true
25  end
```

A.5 Visualization functions

A.5.1 plot_mcmc

Function tha plots the MCMC chains of the selected paramters

Input:

- **B::Array{Float64}**: Array containing the samples of the regressive coefficients
- **Sigma::Array{Float64}**: Array containing the samples of Covariance matrix
- **B_true::Array{Float64}** Array containing the true values of the regressive coefficients
- **Sigma_true::Array{Float64}** Array containing the true values of Sigma
- **R::Array{Float64}** Array containing the samples of rotation matrices
- **R_true::Array{Float64}** Array containing the samples of true rotation matrices
- **theta::Array{Float64}** Array containing the samples of angles
- **theta_true::Array{Float64}** Array containing the samples of true angles

- **plot_flag::Array{Int64}** Array containing flags: 1 to plot the corresponding parameter, 0 otherwise. Order is [B,Sigma,R,Theta].
- **dir::String="Plots/"** Directory where to store the plots: by default, it is "Plots/".

```
function plot_mcmc(B::Array{Float64},Sigma::Array{Float64},B_true::Array{Float64},Sigma_true::Array{Float64},R::Array{Float64},R_true::Array{Float64},theta::Array{Float64},theta_true::Array{Float64},plot_flag::Array{Int64},dir::String="Plots/")
    if isdir(dir) == false
        mkdir(dir)
    else
        rm(dir,recursive=true)
        mkdir(dir)
    end
    I = size(B)[1]
    d = size(B)[2]
    K = size(B)[3]
    p = size(B)[4]

    B_flag = plot_flag[1]
    Sigma_flag = plot_flag[2]
    R_flag = plot_flag[3]
    theta_flag = plot_flag[4]
    if B_flag == 1 || Sigma_flag == 1
        print("Plotting B and Sigma...")
    end
    p_B = Array{Plots.Plot{Plots.GRBackend},1}()
    p_S = Array{Plots.Plot{Plots.GRBackend},1}()


    if B_flag == 1
            if isdir(dir*"Beta/") == false
                mkdir(dir*"Beta/")
            end
            lab = reshape(repeat(["sample";"true"],K*p),1,2*K*p)
            name_B = reshape(["B"*"_"*string(h)*"_"*string(i)*"_"*string(j) for h = 1:d for i =1:K for j = 1:p],1,d*K*p)
            for h = 1:d
                for i = 1:K
                    for j = 1:p
                        m = minimum( [minimum(B[:,h,i,j]) B_true[1,h,i,j] ])-1
                        M = maximum( [maximum(B[:,h,i,j]) B_true[1,h,i,j] ])+1
```

```
35                          p1 = hline!(plot(B[:,h,i,j], size = (1920,1080),
                                legend = true,xlims=(0,I), ylims = (m,M)),[
                                B_true[1,h,i,j]])
36                          push!(p_B, p1)
37                      end
38                  end
39                  p_B_plot = plot(p_B[(h-1)*K*p+1:h*K*p]..., layout = (
                        K,p), title = reshape(name_B[(h-1)*K*p+1:h*K*p],1,
                        K*p), labels = lab)
40                  savefig(p_B_plot,dir*"Beta/Beta_"*string(h)*".png")
41              end
42      end
43
44      if Sigma_flag == 1
45          for i = 1:K
46              for j = 1:K
47                  m = minimum(Sigma[:,i,j])-1
48                  M = maximum(Sigma[:,i,j])+1
49                  p2 = hline!(plot(Sigma[:,i,j], size = (1920,1080), legend =
                        true,xlims = (0,I)),[Sigma_true[i,j]])
50          push!(p_S, p2)
51              end
52          end
53          name_S = reshape(["S"*"_"*string(i)*"_"*string(j) for i =1:K for
                j = 1:K],1,K*K)
54          lab = reshape(repeat(["sample";"true"],K*K),1,2*K*K)
55          p_S = plot(p_S..., layout = K*K, title = name_S, labels = lab)
56          savefig(p_S,dir*"Sigma.png")
57      end
58
59      if B_flag == 1 || Sigma_flag == 1
60          print("Done! \n")
61      end
62
63      if theta_flag == 1
64          print("Plotting angles...")
65          plot_angles(theta,theta_true,dir)
66          print("Done! \n")
67      end
68
69      if R_flag == 1
70          print("Plotting R...")
71          plot_R(R,R_true,dir)
72          print("Done!")
73      end
```

```
74
75 end
```

A.5.2 plot_R

Function that plots the rotations entries

Input:

- **R::Array{Float64}** Array containing the samples of rotation matrices
- **R_true::Array{Float64}** Array containing the samples of true rotation matrices
- **dir::String="Plots/"** Master directory: a folder "R/" will be created inside the specified directory.

```
1  function plot_R(R::Array{Float64},R_true::Array{Float64},dir::String="
       Plots/")
2
3      I = size(R)[1]
4      N = size(R)[2]
5      p = size(R)[3]
6      if isdir(dir*"R/") == false
7          mkdir(dir*"R/")
8      end
9
10     p_R = Array{Plots.Plot{Plots.GRBackend},1}()
11     for s =1:N
12         for i = 1:p
13             for j = 1:p
14             m = minimum(R[:,s,i,j])-1
15             M = maximum(R[:,s,i,j])+1
16             p1 = hline!(plot(R[:,s,i,j], size = (1920,1080),legend = true,
                   xlims = (0,I), ylims = (m,M)),[R_true[s,i,j]])
17             push!(p_R, p1)
18             end
19         end
20     end
21
22     N_p = N*p*p
23     lab = reshape(repeat(["sample";"true"],N_p),1,2*N_p)
```

```
24      name_R = reshape(["R"*"_"*string(s)*"_"*string(i)*"_"*string(j)
            for s =1:N for i =1:p for j = 1:p],1,N_p)
25      for k = 0:N-1
26          p_S = p_R[p*p*k+1:p*p*k+p*p]
27          title_S = reshape(name_R[p*p*k+1:p*p*k+p*p],1,p*p)
28          labels_S = reshape(lab[2*p*p*k+1: 2*p*p+2*p*p*k],1,2*p*p)
29          p_S1 = plot(p_S..., layout = p*p, title = title_S, labels =
                labels_S)
30          savefig(p_S1,dir*"R/R_"*string(k+1)*".png")
31          print("Plot "*string(k+1)*" finished! \n")
32      end
33
34  end
```

A.5.3 plot_angles

Function that plots the angles' chains.

Input:

- **theta::Array{Float64}** Array containing the angles' samples
- **theta_true::Array{Float64}** Array containing true angles' samples
- **dir::String = "Plots/"** Master directory: a folder "theta/" will be created inside the specified directory.

```
1   function plot_angles(theta::Array{Float64},theta_true::Array{Float64},dir
        ::String = "Plots/")
2
3       I = size(theta)[1]
4       N = size(theta)[2]
5       p = size(theta)[3]
6       if isdir(dir*"Theta/") == false
7           mkdir(dir*"Theta/")
8       end
9       p_R = Array{Plots.Plot{Plots.GRBackend},1}()
10      for s =1:N
11          for i = 1:p
12              if i != 2
13                  p1 = hline!(plot(theta[:,s,i], size = (1920,1080),legend =
                        true,ylims=(0,2pi),xlims=(0,I)),[theta_true[s,i]])
14              else
15                  p1 = hline!(plot(theta[:,s,i], size = (1920,1080),legend =
                        true,ylims=(0,pi), xlims = (0,I)),[theta_true[s,i]])
```

```
16                  end
17                  push!(p_R, p1)
18              end
19          end
20
21          N_p = N*p
22          lab = reshape(repeat(["sample";"true"],N_p),1,2*N_p)
23          name_R = reshape(["theta"*"_"*string(s)*"_"*string(i) for s =1:N
                for i =1:3],1,N_p)
24          for k = 0:N-1
25              p_S = p_R[3k+1:3k+3]
26              title_S = reshape(name_R[3k+1:3k+3],1,p)
27              labels_S = reshape(lab[6k+1: 6+6k],1,2*p)
28              p_S1 = plot(p_S..., layout = 3, title = title_S, labels = labels_S)
29              savefig(p_S1,dir*"Theta/theta_"*string(k+1)*".png")
30              println("Plot"*string(k+1)*" finished!")
31          end
32
33  end
```

A.6 Main scripts

Script used to make simple tests over sinthetic datasets

A.6.1 test

```
1   include("mcmc.jl")
2   Random.seed!(0)
3
4   N = 20; #Number of consfigurations
5   K = 10; #Number of landmarks
6   d = 1; #Number of covariates
7   p = 2; # Number of coordinates
8
9   #Only iontercept
10  if d ==1
11      z = repeat([1.0],N) #Matri x of covariates
12  end
13
14  #One covariate
15  if d == 2
16      z = repeat([1.0 0.0],N) #Matrix of covariates
17      for i = 1:N
```

```
18            mu = 1
19            sigma = 1
20            z[i,2] = log(T[i])^2
21        end
22    end
23
24    #Two covariates
25    if d==3
26        z = repeat([1.0 0.0 0.0],N)
27        for i =1:N
28            z[i,2] = rand(Normal(5,1))
29            z[i,3] = rand(Normal(2,1))
30        end
31    end
32
33    #Covariates standardization
34    for h =2:d
35        m[h-1] = mean(z[:,h])
36        v[h-1] = std(z[:,h])
37        z[:,h] .-= m[h-1]
38        z[:,h] /= v[h-1]
39    end
40
41
42    #Desing matrix
43    Z = zeros(N,K,(K)*d)
44    for i =1:N
45        Z[i,:,:] = kron(I(K),z[i,:]')
46    end
47
48    #Variance-Covariance matrix
49    nu = K+1
50    Sigma_true = rand(InverseWishart(nu,Psi))
51    k = 1
52    VarCov = 1.0*kron(I(p),Sigma_true)
53
54
55    #Real regressive coefficients
56    mu1 = rand(Normal(5,1),K*p)
57    mu2 = rand(Normal(0,1),K*p)
58    mu3 = rand(Normal(0,1),K*p)
59
60    B_true = zeros(d,K,p)
61    B_true[1,:,:] = reshape(mu1,(1,K,p))
62
```

```
63 if d==2
64     B_true[2,:,1] = mu2
65 end
66
67 if d == 3
68     B_true[3,:,:] = mu3
69 end
70
71 #Build dataset
72 samples, Y, R_true, theta_true = makedataset(N,d,K,p,z,B_true,VarCov)
       ;
73
74 #MCMC parameters
75 I_max = 30000
76 burn_in = 20000
77 thin = 10
78 original = 0 #Set = 1 to use original samples instead of simulated one
79
80
81 #---- PRIORS PARAMETERS ---#
82 #Sigma
83 nu_prior = K+1;
84 Psi_prior = 1.0*Matrix(1.0*I(K));
85 #Beta
86 M_prior = zeros(p,K*d);
87 #M_prior = reshape(mu,p,d*K)
88 V_prior = 10.0^6*Matrix(I(K*d))
89
90
91
92 #MCMC
93 theta_sim = [1 0 0] #When p = 2 --> [1 0 0]
94 beta_sim = 1
95 Sigma_sim = 1
96
97 plot_flag = [1 1 1 1] #Flags to choose what to plot, order is [B,Sigma,R,
       Theta]
98
99 B, Sigma_est, theta, R, X = mcmc(I_max, burn_in, thin, d,K,p,N,z,Z,Y,
       nu_prior,Psi_prior,M_prior,V_prior, original, samples,B_true,
       Sigma_true, theta_true,theta_sim,beta_sim,Sigma_sim);
100 B_true_tensor = permutedims(reshape(B_true,d,K,p,1),(4,1,2,3))
101 #Identify params
102 samples_id,X_id, B_id, B_true_id, R_id, R_true_id, theta_id,
       theta_true_id =identify_params(Y,B, B_true_tensor, Sigma_est,
```

```
    Sigma_true, R, R_true)

plot_mcmc(B_id,Sigma_est,B_true_id,Sigma_true,R_id,R_true_id,
    theta_id,theta_true_id,plot_flag)

Y_est = zeros(I_max,N,K,p)
for i =1:size(X,1)
    for s =1:N
        Y_est[i,s,:,:] = X[i,s,:,:]*R[i,s,:,:]
    end
end
Y_m = reshape(mean(Y_est,dims=[1,2]),K,p)
Y_m_true = reshape(mean(Y,dims=1),K,p)
rho = Riemann_distance(Y_m,Y_m_true)
```

Bibliography

[1] C. G. Khatri, K. V. Mardia *The von Mises-Fisher Matrix Distribution in Orientation Statistic*, 1977

[2] G. Strang, *Linear algebra and learning from data*, 2019

[3] G. Mastrantonio, A. Di Noia, G. Jona Lasinio, Bayesian shape and size regression modelling

[4] Ian L. Dryden, Alfred Kume, Phillip J. Paine & Andrew T. A. Wood (2021) Regression Modeling for Size-and-Shape Data Based on a Gaussian Model for Landmarks, Journal of the American Statistical Association, 116:534, 1011-1022, DOI: 10.1080/01621459.2020.1724115

[5] J. Down, *Orientation statistics*,Biometrika Vol. 59, No. 3 (Dec., 1972), pp. 665-676 (12 pages)

[6] Jeffrey M. Lee, *Manifolds and differential geometry*, 2009

[7] Kanti V. Mardia, E. Jupp, *Directional statistics*, 2000

[8] Kendall, D. G. (1977). The diffusion of shape. Advances in Applied Probability

[9] Mardia, K. V. (1972). Statistics of Directional Data. Academic Press, London.

[10] L. dryden, Kanti V. Mardia, *Statistical shape analysis with applications in R*, 2012

[11] P. Biscari, T. Ruggeri, G. Saccomandi, M. Vianello, *Meccanica razionale*, 2014