COMPUTO

ISSN 2824-7795

Optimal projection for parametric importance sampling in high dimensions

Maxime El Masri ONERA/DTIS, ISAE-SUPAERO, Université de Toulouse
Jérôme Morio ONERA/DTIS, Université de Toulouse
Florian Simatos ISAE-SUPAERO, Université de Toulouse

Date published: 2023-10-15 Last modified:

Abstract

We propose a dimension reduction strategy in order to improve the performance of importance sampling in high dimensions. The idea is to estimate variance terms in a small number of suitably chosen directions. We first prove that the optimal directions, i.e., the ones that minimize the Kullback–Leibler divergence with the optimal auxiliary density, are the eigenvectors associated with extreme (small or large) eigenvalues of the optimal covariance matrix. We then perform extensive numerical experiments showing that as dimension increases, these directions give estimations which are very close to optimal. Moreover, we demonstrate that the estimation remains accurate even when a simple empirical estimator of the covariance matrix is used to compute these directions. The theoretical and numerical results open the way for different generalizations, in particular the incorporation of such ideas in adaptive importance sampling schemes.

Keywords: Rare event simulation, Parameter estimation, Importance sampling, Dimension reduction, Kullback–Leibler divergence, Projection

Contents

1 Introduction

Importance Sampling (IS) is a stochastic method to estimate integrals of the form $\mathscr{E} = \int \phi(\mathbf{x}) f(\mathbf{x}) \mathrm{d}\mathbf{x}$ with a black-box function ϕ and a probability density function (pdf) f. It rests upon the choice of an auxiliary density which can significantly improve the estimation compared to the naive Monte Carlo (MC) method (Agapiou et al. 2017), (Owen and Zhou 2000). The theoretical optimal IS density, also called zero-variance density, is defined by $\phi f / \mathscr{E}$ when ϕ is a positive function. This density is not available in practice as it involves the unknown integral $\mathscr{E}$, but a classical strategy consists in searching for an optimal approximation in a parametric family of densities. By minimising a "distance" to the optimal IS density, such as the Kullback–Leibler divergence, one can find optimal parameters in this family to get an efficient sampling pdf. Adaptive Importance Sampling (AIS) algorithms, such as the Mixture Population Monte Carlo method (Cappé et al. 2008), the Adaptive Multiple Importance Sampling method (Cornuet et al. 2012), or the Cross Entropy method (Rubinstein and Kroese 2011a), estimate the optimal parameters adaptively by updating at intermediate levels (Bugallo et al. 2017).

These techniques work very well, but only for moderate dimensions. In high dimensions, most of these techniques fail to give suitable parameters for two reasons:

1. the weight degeneracy problem, for which the self-normalized likelihood ratios (weights) in the IS densities degenerate in the sense that the largest one takes all the mass, while all other weights are negligible so that the final estimation essentially uses only one sample. See for instance (Bengtsson, Bickel, and Li 2008) for a theoretical analysis in the related context of particle filtering. The conditions under which importance sampling is applicable in high dimensions are notably investigated in a reliability context in (Au and Beck 2003): it is remarked that the optimal covariance matrix should not deviate significantly from the identity matrix. (El-Laham, Elvira, and Bugallo 2019) tackle the weight degeneracy problem by applying a

recursive shrinkage of the covariance matrix, which is constructed iteratively with a weighted sum of the sample covariance estimator and a biased, but more stable, estimator;

2. the intricate estimation of distribution parameters in high dimensions and particularly covariance matrices, whose size increases quadratically in the dimension (Ashurbekova et al. 2020),(Ledoit and Wolf 2004). Empirical covariance matrix estimate has notably a slow convergence rate in high dimensions (Fan, Fan, and Lv 2008). For that purpose, dimension reduction techniques can be applied. The idea was recently put forth to reduce the effective dimension by only estimating these parameters (in particular the covariance matrix) in suitable directions (El Masri, Morio, and Simatos 2021), (Uribe et al. 2021). In this paper we delve deeper into this idea.

The main contribution of the present paper is to identify the optimal directions in the fundamental case when the parametric family is Gaussian, and perform numerical simulations in order to understand how they behave in practice. In particular, we propose directions which, in contrast to the recent paper (Uribe et al. 2021), do not require the objective function to be differentiable, and moreover optimizes the Kullback–Leibler distance with the optimal density instead of simply an upper bound on it, as in (Uribe et al. 2021). In Section 3.1 we elaborate in more details on the differences between the two approaches.

The paper is organised as follows: in Section 2 we recall the foundations of IS. In Section 3, we state our main theoretical result and we compare it with the current state-of-the-art. The proof of our theoretical result are given in Appendix; Section 4 introduces the numerical framework that we have adopted, and Section 5 presents the numerical results obtained on five different test cases to assess the efficiency of the directions that we propose. We conclude in Section 6 with a summary and research perspectives.

2 Importance Sampling

We consider the problem of estimating the following integral:

$$\mathscr{E} = \mathbb{E}_f(\phi(\mathbf{X})) = \int \phi(\mathbf{x}) f(\mathbf{x}) \mathrm{d}\mathbf{x},$$

where $\mathbf{X}$ is a random vector in $\mathbb{R}^n$ with standard Gaussian pdf f, and $\phi : \mathbb{R}^n \to \mathbb{R}_+$ is a real-valued, non-negative function. The function ϕ is considered as a black-box function which is potentially expensive to evaluate, and this means that the number of calls to ϕ should be limited.

IS is an approach used to reduce the variance of the classical Monte Carlo estimator of $\mathscr{E}$. The idea of IS is to generate a random sample $\mathbf{X}_1, \dots, \mathbf{X}_N$ from an auxiliary density g, instead of f, and to compute the following estimator:

$$\widehat{\mathscr{E}_N} = \frac{1}{N} \sum_{i=1}^{N} \phi(\mathbf{X}_i) L(\mathbf{X}_i), \tag{1}$$

with $L = f/g$ the likelihood ratio, or importance weight, and the auxiliary density g, also called importance sampling density, is such that $g(\mathbf{x}) = 0$ implies $\phi(\mathbf{x}) f(\mathbf{x}) = 0$ for every $\mathbf{x}$ (which makes the product ϕL well-defined). This estimator is consistent and unbiased but its accuracy strongly depends on the choice of the auxiliary density g. It is well known that the optimal choice for g is (Bucklew 2013)

$$g^*(\mathbf{x}) = \frac{\phi(\mathbf{x}) f(\mathbf{x})}{\mathscr{E}}, \ \mathbf{x} \in \mathbb{R}^n.$$

Indeed, for this choice we have $\phi L = \mathscr{E}$ and so $\widehat{\mathscr{E}_N}$ is actually the deterministic estimator $\mathscr{E}$. For this reason, g^* is sometimes called zero-variance density, a terminology that we will adopt here.

Of course, g^* is only of theoretical interest as it depends on the unknown integral $\mathscr{E}$. However, it gives an idea of good choices for the auxiliary density g, and we will seek to approximate g^* by an auxiliary density that minimizes a distance between g^* and a given parametric family of densities.

In this paper, the parametric family of densities is the Gaussian family $\{g_{\mathbf{m},} : \mathbf{m} \in \mathbb{R}^n, \in \mathcal{S}_n^+\}$, where $g_{\mathbf{m},}$ denotes the Gaussian density with mean $\mathbf{m} \in \mathbb{R}^n$ and covariance matrix $\in \mathcal{S}_n^+$ with $\mathcal{S}_n^+ \subset \mathbb{R}^{n\times n}$ the set of symmetric, positive-definite matrices:

$$g_{\mathbf{m},}(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}||^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{m})^{\top -1}(\mathbf{x}-\mathbf{m})\right), \ \mathbf{x} \in \mathbb{R}^n.$$

with $||$ the determinant of . Moreover, we will consider the Kullback–Leibler (KL) divergence to measure a "distance" between g^* and $g_{\mathbf{m},}$. Recall that for two densities f and h, with f absolutely continuous with respect to h, the KL divergence $D(f,h)$ between f and h is defined by:

$$D(f,h) = \mathbb{E}_f\left[\log\left(\frac{f(\mathbf{X})}{h(\mathbf{X})}\right)\right] = \int \log\left(\frac{f(\mathbf{x})}{h(\mathbf{x})}\right) f(\mathbf{x})\mathrm{d}\mathbf{x}.$$

Thus, our goal is to approximate g^* by $g_{\mathbf{m}^*,^*}$ with the optimal mean vector $\mathbf{m}^*$ and the optimal covariance matrix * given by:

$$(\mathbf{m}^*, ^*) = \arg\min\left\{D(g^*, g_{\mathbf{m},}) : \mathbf{m} \in \mathbb{R}^n, \in \mathcal{S}_n^+\right\}. \tag{2}$$

This optimization is in general convex and differentiable with respect to $\mathbf{m}$ and . Moreover, the solution of Equation 2 can be computed analytically by cancelling the gradient. In the Gaussian case, it is thus proved that $\mathbf{m}^*$ and * are simply the mean and variance of the zero-variance density (Rubinstein and Kroese 2011b), (Rubinstein and Kroese 2017a):

$$\mathbf{m}^* = \mathbb{E}_{g^*}(\mathbf{X}) \qquad \text{and} \qquad ^* = \mathrm{Var}_{g^*}(\mathbf{X}). \tag{3}$$

3 Efficient dimension reduction

3.1 Projecting onto a low-dimensional subspace

As g^* is unknown, the optimal parameters $\mathbf{m}^*$ and * given by Equation 3 are not directly computable. However, we can sample from the optimal density as it is known up to a multiplicative constant. Therefore, usual estimation schemes start with estimating $\mathbf{m}^*$ and *, say through $\widehat{\mathbf{m}}^*$ and$\widehat{}^*$, respectively, and then use these approximations to estimate $\mathscr{E}$ through Equation 1 with the auxiliary density $g_{\widehat{\mathbf{m}}^*,\widehat{}^*}$. Although the estimation of $\mathscr{E}$ with the auxiliary density $g_{\mathbf{m}^*,^*}$ usually provides very good results, it is well-known that in high dimensions, the additional error induced by the estimations of $\mathbf{m}^*$ and * severely degrades the accuracy of the final estimation (Papaioannou, Geyer, and Straub 2019), (Uribe et al. 2021). The main problem lies in the estimation of * which, in dimension n, involves the estimation of a quadratic (in the dimension) number of terms, namely $n(n+1)/2$. Recently, the idea to overcome this problem by only evaluating variance terms in a small number of influential directions was explored in (El Masri, Morio, and Simatos 2021) and (Uribe et al. 2021). In these two papers, the auxiliary covariance matrix is modeled in the form

$$= \sum_{i=1}^{k}(\nu_i - 1)\mathbf{d}_i\mathbf{d}_i^\top + I_n \tag{4}$$

where the $\mathbf{d}_i$'s are the k orthonormal directions which are deemed influential. It is easy to check that is the covariance matrix of the Gaussian vector

$$\nu_1^{1/2}Y_1\mathbf{d}_1 + \cdots + \nu_k^{1/2}Y_k\mathbf{d}_k + Y_{k+1}\mathbf{d}_{k+1} + \cdots + Y_n\mathbf{d}_n$$

where the Y_i's are i.i.d. standard normal random variables (one-dimensional), and the $n-k$ vectors $(\mathbf{d}_{k+1}, \dots, \mathbf{d}_n)$ complete $(\mathbf{d}_1, \dots, \mathbf{d}_k)$ into an orthonormal basis. In particular, v_i is the variance in the direction of $\mathbf{d}_i$, i.e., $v_i = \mathbf{d}_i^\top \mathbf{d}_i$. In Equation 4, k can be considered as the effective dimension in which variance terms are estimated. In other words, in (El Masri, Morio, and Simatos 2021) and (Uribe et al. 2021), the optimal variance parameter is not sought in $\mathcal{S}_n^+$ as in Equation 2, but rather in the subset of matrices of the form

$$\mathscr{L}_{n,k} = \left\{ \sum_{i=1}^{k} (\alpha_i - 1) \frac{\mathbf{d}_i \mathbf{d}_i^\top}{\|\mathbf{d}_i\|^2} + I_n \ : \ \alpha_1, \dots, \alpha_k > 0 \ \text{ and the } \mathbf{d}_i\text{'s are orthogonal} \right\}.$$

The relevant minimization problem thus becomes

$$(\mathbf{m}_k^*, {}_k^*) = \arg\min \left\{ D(g^*, g_{\mathbf{m},}) \ : \ \mathbf{m} \in \mathbb{R}^n, \ \ \in \mathscr{L}_{n,k} \right\} \tag{5}$$

instead of Equation 2, with the effective dimension k being allowed to be adjusted dynamically. By restricting the space in which the variance is assessed, one seeks to limit the number of variance terms to be estimated. The idea is that if the directions are suitably chosen, then the improvement of the accuracy due to the smaller error in estimating the variance terms will compensate the fact that we consider less candidates for the covariance matrix. In (El Masri, Morio, and Simatos 2021), the authors consider $k = 1$ and $\mathbf{d}_1 = \mathbf{m}^*/\|\mathbf{m}^*\|$. When f is Gaussian, this choice is motivated by the fact that, due to the light tail of the Gaussian random variable and the reliability context, the variance should vary significantly in the direction of $\mathbf{m}^*$ and so estimating the variance in this direction can bring information. In Section 3.5, we use the techniques of the present paper to provide a stronger theoretical justification of this choice, see Theorem 3.2 and the discussion following it. The method in (Uribe et al. 2021) is more involved: k is adjusted dynamically, while the directions $\mathbf{d}_i$ are the eigenvectors associated to the largest eigenvalues of a certain matrix. They span a low-dimensional subspace called Failure-Informed Subspace, and the authors in (Uribe et al. 2021) prove that this choice minimizes an upper bound on the minimal KL divergence. In practice, this algorithm yields very accurate results. However, we will not consider it further in the present paper for two reasons. First, this algorithm is tailored for the reliability case where $\phi = \mathbb{I}_{\{\varphi \geq 0\}}$, with a function $\varphi : \mathbb{R}^n \to \mathbb{R}$, whereas our method is more general and applies to the general problem of estimating an integral (see for instance our test case of Section 5.5). Second, the algorithm in (Uribe et al. 2021) requires the evaluation of the gradient of the function φ. However, this gradient is not always known and can be expensive to evaluate in high dimensions; in some cases, the function φ is even not differentiable, as will be the case in our numerical example in Section 5.4. In contrast, our method makes no assumption on the form or smoothness of ϕ: it does not need to assume that it is of the form $\mathbb{I}_{\{\varphi \geq 0\}}$, or to assume that $\nabla\varphi$ is tractable. For completeness, whenever the algorithm of (Uribe et al. 2021) was applicable and computing the gradient of φ did not require any additional simulation budget, we have run it on the test cases considered here and found that it outperformed our algorithm. In more realistic settings, computing $\nabla\varphi$ would likely increase the simulation budget, and it would be interesting to compare the two algorithms in more details to understand when this extra computation cost is worthwhile. We reserve such a question for future research and will not consider the algorithm of (Uribe et al. 2021) further, as our aim in this paper is to establish benchmark results for a general algorithm which works for any function ϕ.

3.2 Definition of the function ℓ

The statement of our result involves the following function ℓ, which is represented in Figure 1:

$$\ell : x \in (0, \infty) \mapsto -\log(x) + x - 1. \tag{6}$$

In the following, $(\lambda, \mathbf{d}) \in \mathbb{R} \times \mathbb{R}^n$ is an eigenpair of a matrix A if $A\mathbf{d} = \lambda\mathbf{d}$ and $\|\mathbf{d}\| = 1$. A diagonalizable matrix has n distinct eigenpairs, say $((\lambda_i, \mathbf{d}_i), i = 1, \dots, n)$, and we say that these eigenpairs are ranked

in decreasing ℓ-order if $\ell(\lambda_1) \geq \cdots \geq \ell(\lambda_n)$. In the rest of the article, we denote as $(\lambda_i^*, \mathbf{d}_i^*)$ the eigenpairs of * ranked in decreasing ℓ-order and as $(\hat{\lambda}_i^*, \hat{\mathbf{d}}_i^*)$ the eigenpairs of $\hat{}^*$ ranked in decreasing ℓ-order.

```
##########################################################################
# Figure 1. Plot of the function "l"
##########################################################################
import numpy as np
import scipy as sp
import pickle
import scipy.stats
import matplotlib.pyplot as plt

### the following library is available on the following website :
### "Papaioannou, I., Geyer, S., and Straub, D. (2019b).
### Software tools for reliability analysis :
### Cross entropy method and improved cross entropy method. Retrieved from
### https://www.cee.ed.tum.de/en/era/software/reliability/"
from CEIS_vMFNM import *

from IPython.display import display, Math, Latex
from IPython.display import Markdown
from tabulate import tabulate
np.random.seed(10)

x = np.linspace(np.finfo(float).eps,4.0,100)
y = -np.log(x) + x -1

# plot
fig, ax = plt.subplots()
ax.plot(x, y, linewidth=2.0)
ax.set(xlim=(0, 4), xticks=[0,1,2,3],
       ylim=(0, 0.5), yticks=[0,0.5,1,1.5])
plt.grid()
plt.xlabel(r"$x$",fontsize=16)
plt.ylabel(r"$\ell(x)$",fontsize=16)
for tickLabel in plt.gca().get_xticklabels() + plt.gca().get_yticklabels():
    tickLabel.set_fontsize(16)
plt.show()
```

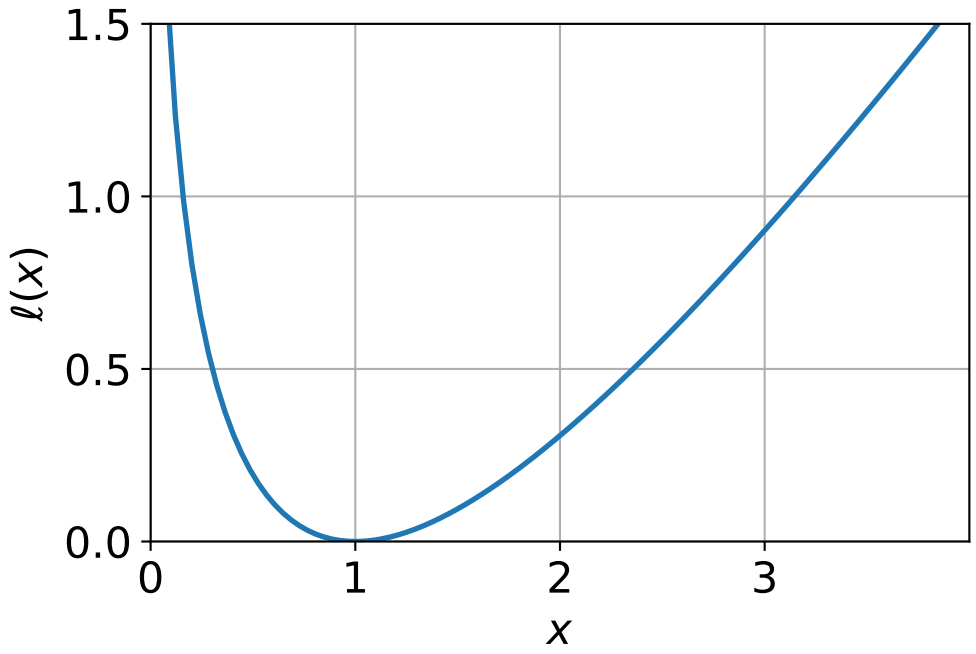


Figure 1: Plot of the function ℓ given by Equation 6.

3.3 Main result of the paper

The main result of the present paper is to compute the exact value for $\overset{*}{k}$ in Equation 5, which therefore paves the way for efficient high-dimensional estimation schemes.

Theorem 3.1. *Let $(\lambda_i^*, \mathbf{d}_i^*)$ be the eigenpairs of $\overset{*}{}$ ranked in decreasing ℓ-order. Then for $1 \leq k \leq n$, the solution $(\mathbf{m}_k^*, \overset{*}{k})$ to Equation 5 is given by*

$$\mathbf{m}_k^* = \mathbf{m}^* \quad \textit{and} \quad \overset{*}{k} = I_n + \sum_{i=1}^{k} \left(\lambda_i^* - 1\right) \mathbf{d}_i^* (\mathbf{d}_i^*)^\top. \tag{7}$$

The proof of Theorem 3.1 is detailed in Appendix A. For $k = 1$ for instance, the matrix $\overset{*}{1} = I_n + (\lambda_1^* - 1)\mathbf{d}_1^*(\mathbf{d}_1^*)^\top$ with $(\lambda_1^*, \mathbf{d}_1^*)$ the eigenpair of $\overset{*}{}$ such as λ_1^* is either the largest or the smallest eigenvalue of $\overset{*}{}$, depending on which one maximizes ℓ.

This theoretical result therefore suggests to reduce dimension by computing the covariance matrix$\hat{\overset{**}{}}$ and its eigenpairs, rank them in decreasing ℓ-order and then use the k first eigenpairs $((\hat{\lambda}_i^*, \hat{\mathbf{d}}_i^*), i = 1, \dots, k)$ to build the covariance matrix$\hat{\overset{*}{k}} = \sum_{i=1}^{k}(\hat{\lambda}_i^* - 1)\hat{\mathbf{d}}_i^*(\hat{\mathbf{d}}_i^*)^\top + I_n$ and the corresponding auxiliary density. This scheme is summarized in Algorithm 1. The effective dimension k is obtained by Algorithm 2, see Section 3.4 below. The proof of the theorem is shown in Appendix A.

Remark. Since the function ℓ is minimized at 1, eigenpairs with $\lambda_i^* = 1$ are selected in the sum of Equation 7 once all other eigenpairs have been picked as the eigenpairs are ℓ-ordered: in other words, if $\lambda_i^* = 1$ then $\lambda_j^* = 1$ for all $j \geq i$. Note also that the minimizer 1 plays a special role as we are interested in covariance matrices of $\mathscr{L}_{n,k}$ which, once diagonalized, have mostly ones in the main diagonal (except for k values associated with the α_i). As k will be small (See Section 3.4), typically $k = 1$ or 2, this amounts to finding covariance matrices that are perturbations of the identity (this is relevant as we assume f is standard Gaussian). Therefore, when approximating $\overset{*}{}$ by such matrices, we should first consider eigenvalues as different as possible from 1 (with the discrepancy from 1 being measured by ℓ).

Algorithm 1 Algorithm suggested by Theorem 1.

Data: Sample sizes N and M
Result: Estimation $\widehat{\mathscr{E}_N}$ of integral $\mathscr{E}$
- Generate a sample $\mathbf{X}_1^*, \dots, \mathbf{X}_M^*$ on $\mathbb{R}^n$ independently according to g^*
- Estimate $\widehat{\mathbf{m}}^*$ and $\hat{}^*$ defined in Equation 8 and Equation 9 with this sample
- Compute the eigenpairs $(\hat{\lambda}_i^*, \hat{\mathbf{d}}_i^*)$ of $\hat{}^*$ ranked in decreasing ℓ-order
- Compute the matrix $\hat{}_k^* = \sum_{i=1}^k (\hat{\lambda}_i^* - 1)\hat{\mathbf{d}}_i^* (\hat{\mathbf{d}}_i^*)^\top + I_n$ with k obtained by applying Algorithm 2 with input $(\hat{\lambda}_1^*, \dots, \hat{\lambda}_n^*)$
- Generate a new sample $\mathbf{X}_1, \dots, \mathbf{X}_N$ independently from $g' = g_{\widehat{\mathbf{m}}^*, \hat{}_k^*}$
- Return $\widehat{\mathscr{E}_N} = \frac{1}{N}\sum_{i=1}^N \phi(\mathbf{X}_i)\frac{f(\mathbf{X}_i)}{g'(\mathbf{X}_i)}$

In the first step of Algorithm 1, we assume g^* can be sampled independently. This is a reasonable assumption as classical techniques such as importance sampling with self-normalized weights or Markov Chain Monte Carlo (MCMC) can be applied in this case (see for instance (Chan and Kroese 2012), (Grace, Kroese, and Sandmann 2014)). In this paper, we choose to apply a basic rejection method that yields perfect independent samples from g^*, possibly at the price of a high computational cost. As the primary goal of this paper is to understand whether the $\mathbf{d}_i^*$'s are indeed good projection directions, this cost will not be taken into account. Possible improvements to relax this assumption are discussed in the conclusion of the paper and in Appendix C.

3.4 Choice of the number of dimensions k

The choice of the effective dimension k, i.e., the number of projection directions considered, is important. If it is close to n, then the matrix $\hat{}_k^*$ will be close to $\hat{}^*$ which is the situation we want to avoid in the first place. On the other hand, setting $k = 1$ in all cases may be too simple and lead to suboptimal results. In practice, however this is often a good choice. In order to adapt k dynamically, we consider a simple method based on the value of the KL divergence. Given the eigenvalues $\lambda_1, \dots, \lambda_n$ ranked in decreasing ℓ-order, we look for the maximal gap between two consecutive eigenvalues of the sequence $(\ell(\lambda_1), \dots, \ell(\lambda_n))$. This allows to choose k such that $\sum_{i=1}^k \ell(\lambda_i)$ is close to $\sum_{i=1}^n \ell(\lambda_i)$ which is equal, up to an additive constant, to the minimal KL divergence (shown in Lemma 6.1). The precise method is described in Algorithm 2.

Algorithm 2 Choice of the number of dimensions

Data: Sequence of positive numbers $\lambda_1, \dots, \lambda_n$ in decreasing ℓ-order
Result: Number of selected dimensions k
- Compute the increments $\delta_i = \ell(\lambda_{i+1}) - \ell(\lambda_i)$ for $i = 1 \dots n-1$
- Return $k = \arg\max \delta_i$, the index of the maximum of the differences.

3.5 Theoretical result concerning the projection on $\mathbf{m}^*$

In (El Masri, Morio, and Simatos 2021), the authors propose to project on the mean $\mathbf{m}^*$ of the optimal auxiliary density g^*. Numerically, this algorithm is shown to perform well, but only a very heuristic explanation based on the light tail of the Gaussian distribution is provided to motivate this choice. It turns out that the techniques used in the proof of Theorem 3.1 can shed light on why projecting on $\mathbf{m}^*$ may indeed be a good idea. Let us first state our theoretical result, and then explain why it justifies the idea of projecting on $\mathbf{m}^*$.

Theorem 3.2. *Consider* $\in \mathscr{L}_{n,1}$ *of the form* $= I_n + (\alpha - 1)\mathbf{d}\mathbf{d}^\top$ *with* $\alpha > 0$ *and* $\|\mathbf{d}\| = 1$. *Then the minimizer in* $(\alpha, \mathbf{d})$ *of the KL divergence between* f *and* $g_{\mathbf{m}^*,}$ *is* $(1 + \|\mathbf{m}^*\|^2, \mathbf{m}^*/\|\mathbf{m}^*\|)$*:*

$$\left(1 + \|\mathbf{m}^*\|^2, \mathbf{m}^*/\|\mathbf{m}^*\|\right) = \arg\min_{\alpha,\mathbf{d}} \left\{ D(f, g_{\mathbf{m}^*, I_n + (\alpha-1)\mathbf{d}\mathbf{d}^\top}) \,:\, \alpha > 0,\ \|\mathbf{d}\| = 1 \right\}.$$

The proof of Theorem 3.2 is detailed in Appendix A. In other words, $\mathbf{m}^*$ appears as an optimal projection direction when one seeks to minimize the KL divergence between f and the Gaussian density with mean $\mathbf{m}^*$ and covariance of the form $I_n + (\alpha - 1)\mathbf{d}\mathbf{d}^\top$. Let us now explain why this minimization problem is indeed relevant, and why choosing an auxiliary density which minimizes this KL divergence may indeed lead to an accurate estimation. The justification deeply relies on the recent results by (Chatterjee and Diaconis 2018).

As mentioned above, in a reliability context where one seeks to estimate a small probability $p = \mathbb{P}(\mathbf{X} \in A)$, Theorem 1.3 in (Chatterjee and Diaconis 2018) shows that $D(g^*, g)$ governs the sample size required for an accurate estimation of p: more precisely, the estimation is accurate if the sample size is larger than $e^{D(g^*,g)}$, and inaccurate otherwise. This motivates the rationale for minimizing the KL divergence with g^*.

However, in high dimensions, importance sampling is known to fail because of the weight degeneracy problem whereby $\max_i L_i / \sum_i L_i \approx 1$, with the L_i's the unnormalized importance weights, or likelihood ratios: $L_i = f(\mathbf{X}_i)/g(\mathbf{X}_i)$ with the $\mathbf{X}_i$'s i.i.d. drawn according to g. Theorem 2.3 in (Chatterjee and Diaconis 2018) shows that the weight degeneracy problem is avoided if the empirical mean of the likelihood ratios is close to 1, and for this, Theorem 1.1 in (Chatterjee and Diaconis 2018) shows that the sample size should be larger than $e^{D(f,g)}$. In other words, these results suggest that the KL divergence with g^* governs the sample size for an accurate estimation of p, while the KL divergence with f governs the weight degeneracy problem.

In light of these results, it becomes natural to consider the KL divergence with f and not only g^* (Owen and Zhou 2000). Of course, minimizing $D(f, g_{\mathbf{m},})$ without constraints on $\mathbf{m}$ and is trivial since $g_{\mathbf{m},} = f$ for $\mathbf{m} = 0$ and $= I_n$. However, these choices are the ones we want to avoid in the first place, and so it makes sense to impose some constraints on $\mathbf{m}$ and . If one keeps in mind the other objective of getting close to g^*, then the choice $\mathbf{m} = \mathbf{m}^*$ becomes very natural, and we are led to considering the optimization problem of Theorem 3.2 (when $\in \mathscr{L}_{n,1}$ is a rank-1 perturbation of the identity).

4 Computational framework

4.1 Numerical procedure for IS estimate comparison

The objective of the numerical simulations is to evaluate the impact of the choice of the covariance matrix on the estimation accuracy of a high dimensional integral $\mathscr{E}$. We thus want to compare the IS estimation results for different auxiliary densities and more particularly for different choices of the auxiliary covariance matrix when the IS auxiliary density is Gaussian. The details of the considered covariance matrices is given in Section 4.2. To extend this comparison, we also compute the results when the IS auxiliary density is chosen with the von Mises–Fisher–Nakagami (vMFN) model recently proposed in (Papaioannou, Geyer, and Straub 2019) for high dimensional probability estimation (See Appendix B).

In Section 5 we test these different models of auxiliary densities on five test cases, where f is a standard Gaussian density. This choice is not a theoretical limitation as we can in principle always come back to this case by transforming the vector $\mathbf{X}$ with isoprobabilistic transformations (see for instance (Hohenbichler and Rackwitz 1981), (Liu and Der Kiureghian 1986)).

The precise numerical framework that we will consider to assess the efficiency of the different auxiliary models is as follows. We assume first that M i.i.d. random samples $\mathbf{X}_1^*, \dots, \mathbf{X}_M^*$ distributed from g^* are available from rejection sampling (unless in Appendix C where we consider MCMC). From these samples, the parameters of the Gaussian and of the vMFN auxiliary density are computed to get an auxiliary density g'. Finally, N samples are generated from g' to provide an estimation of $\mathscr{E}$ with IS. This procedure is summarized by the following stages:

1. Generate a sample $\mathbf{X}_1^*, \dots, \mathbf{X}_M^*$ independently according to g^*;
2. From $\mathbf{X}_1^*, \dots, \mathbf{X}_M^*$, compute the parameters of the auxiliary parametric density g';
3. Generate a new sample $\mathbf{X}_1, \dots, \mathbf{X}_N$ independently from g';
4. Estimate $\mathscr{E}$ with $\widehat{\mathscr{E}_N} = \frac{1}{N}\sum_{i=1}^{N} \phi(\mathbf{X}_i)\frac{f(\mathbf{X}_i)}{g'(\mathbf{X}_i)}$.

The number of samples M and N are respectively set to $M = 500$ and $N = 2000$. The computational cost to generate $M = 500$ samples distributed from g^* with rejection sampling is often unaffordable in practice; if $\mathscr{E}$ is a probability of order 10^{-p}, then approximately 500×10^p calls to ϕ are necessary for the generation of $\mathbf{X}_1^*, \dots, \mathbf{X}_M^*$. Finally, whatever the auxiliary parametric density g' computed from $\mathbf{X}_1^*, \dots, \mathbf{X}_M^*$, the number of calls to ϕ for the estimation step stays constant and equal to N. The number of calls to ϕ for the whole procedure on a 10^{-p} probability estimation is about $500 \times 10^p + N$. A more realistic situation is considered in Appendix C where MCMC is applied to generate samples from g^*. The resulting samples are dependent but the computational cost is significanlty reduced. The number of calls to ϕ with MCMC is then equal to M which leads to a total computational cost of $M + N$ for the whole procedure.

This procedure is then repeated 500 times to provide a mean estimation $\widehat{\mathscr{E}}$ of $\mathscr{E}$. In the result tables, for each auxiliary density g' we report the corresponding value for the relative error $\widehat{\mathscr{E}}/\mathscr{E} - 1$ and the coefficient of variation of the 500 iterations (the empirical standard deviation divided by $\mathscr{E}$). As was established in the proof of Theorem 3.1, the KL divergence is, up to an additive constant, equal to $D'() = \log|| + \mathrm{tr}(^{*-1})$ which we will refer to as partial KL divergence. In the result tables, we also report thus the mean value of $D'()$ to analyse the relevance of the auxiliary density $g_{\widehat{\mathbf{m}}^*,}$ for six choices of covariance matrix . The next sections specify the different parameters of g' for the Gaussian model and for the vMFN model we have considered in the simulations.

4.2 Choice of the auxiliary density g' for the Gaussian model

The goal is to get benchmark results to assess whether one can improve estimations of Gaussian IS auxiliary density by projecting the covariance matrix * in the proposed directions $\mathbf{d}_i^*$. The algorithm that we study here (Algorithms 1+2) aims more precisely at understanding whether:

- projecting can improve the situation with respect to the empirical covariance matrix;
- the $\mathbf{d}_i^*$'s are good candidates, in particular compared to the choice $\mathbf{m}^*$ suggested in (El Masri, Morio, and Simatos 2021);
- what is the impact in making errors in estimating the eigenpairs $(\lambda_i^*, \mathbf{d}_i^*)$.

Let us define the estimate $\widehat{\mathbf{m}}^*$ of $\mathbf{m}^*$ from the M i.i.d. random samples $\mathbf{X}_1^*, \dots, \mathbf{X}_M^*$ distributed from g^* with

$$\widehat{\mathbf{m}}^* = \frac{1}{M}\sum_{i=1}^{M} \mathbf{X}_i^*. \tag{8}$$

In our numerical test cases, we will compare six different choices of Gaussian auxiliary distributions g' with mean $\widehat{\mathbf{m}}^*$ and the following covariance matrices summarized in Table 1:

1. *: the optimal covariance matrix given by Equation 3;

2. $\hat{}^*$: the empirical estimation of * given by

$$\hat{}^* = \frac{1}{M}\sum_{i=1}^{M}(\mathbf{X}_i^* - \widehat{\mathbf{m}}^*)(\mathbf{X}_i^* - \widehat{\mathbf{m}}^*)^\top. \tag{9}$$

The four other covariance matrices considered in the numerical simulations are of the form $\sum_{i=1}^{k}(\nu_i - 1)\mathbf{d}_i\mathbf{d}_i^\top + I_n$ where ν_i is the variance of$\hat{}^*$ in the direction $\mathbf{d}_i$, $\nu_i = \mathbf{d}_i^\top\hat{}^*\mathbf{d}_i$. The considered choice of k and $\mathbf{d}_i$ gives the following covariance matrices:

3. $\hat{}_{\text{opt}}$ is obtained by choosing $\mathbf{d}_i = \mathbf{d}_i^*$ of Theorem 3.1, which is supposed to be perfectly known from * and k is computed with Algorithm 2;
4. $\hat{}_{\text{opt}}^{+\text{d}}$ is obtained by choosing $\mathbf{d}_i = \hat{\mathbf{d}}_i^*$ the i-th eigenvector of$\hat{}^*$ (in ℓ-order), which is an estimation of $\mathbf{d}_i^*$, and k is computed with Algorithm 2;
5. $\hat{}_{\text{mean}}$ is obtained by choosing $k = 1$ and $\mathbf{d}_1 = \mathbf{m}^*/\|\mathbf{m}^*\|$;
6. $\hat{}_{\text{mean}}^{+\text{d}}$ is obtained by choosing $k = 1$ and $\mathbf{d}_1 = \widehat{\mathbf{m}}^*/\|\widehat{\mathbf{m}}^*\|$, where $\widehat{\mathbf{m}}^*$ given by Equation 8.

The matrices$\hat{}_{\text{opt}}$ and$\hat{}_{\text{mean}}$ use the estimation$\hat{}^*$ with the optimal directions $\mathbf{d}_i^*$ or $\mathbf{m}^*$, while the matrices$\hat{}_{\text{opt}}^{+\text{d}}$ and$\hat{}_{\text{mean}}^{+\text{d}}$ involve an estimation of these directions from$\hat{}^*$. By definition, * will give optimal results, while results for$\hat{}^*$ will deteriorate as the dimension increases, which is the well-known behavior which we try to improve. Moreover, * and the projection directions $\mathbf{d}_i^*$ or $\mathbf{m}^*$, are of course unknown in practice. For simulation comparison purpose, they could be determined analytically in simple test cases and otherwise we obtained them by a brute force Monte Carlo scheme with a very high simulation budget. Finally, we emphasize that Algorithm 1 corresponds to estimating and projecting on the $\mathbf{d}_i^*$'s, and so the matrix$\hat{}_k^*$ of Algorithm 1 is equal to the matrix$\hat{}_{\text{opt}}^{+\text{d}}$.

Table 1: Presentation of the six covariance matrices considered in the numerical examples.

	*	$\hat{}^*$	$\hat{}_{\text{opt}}$	$\hat{}_{\text{mean}}$	$\hat{}_{\text{opt}}^{+\text{d}}$	$\hat{}_{\text{mean}}^{+\text{d}}$
Initial covariance matrix	*	$\hat{}^*$	$\hat{}^*$	$\hat{}^*$	$\hat{}^*$	$\hat{}^*$
Projection directions (exact or estimated)	-	-	Exact	Exact	Esti-mated	Esti-mated
Choice for the projection direction	None	None	Opt	Mean	Opt	Mean

5 Numerical results on five test cases

The proposed numerical framework is applied on three examples that are often considered to assess the performance of importance sampling algorithms and also two test cases from the area of financial mathematics.

5.1 Test case 1: one-dimensional optimal projection

We consider a test case where all computations can be made exactly. This is a classical example of rare event probability estimation, often used to test the robustness of a method in high dimensions.

It is given by $\phi(\mathbf{x}) = \mathbb{I}_{\{\varphi(\mathbf{x})\geq 0\}}$ with φ the following affine function:

$$\varphi \,:\, \mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n \mapsto \sum_{j=1}^{n} x_j - 3\sqrt{n}. \tag{10}$$

The quantity of interest $\mathscr{E}$ is defined as $\mathscr{E} = \int_{\mathbb{R}^n} \phi(\mathbf{x}) f(\mathbf{x}) \mathrm{d}\mathbf{x} = \mathbb{P}_f(\varphi(\mathbf{X}) \geq 0) \simeq 1.35 \cdot 10^{-3}$ for all n where the density f is the standard n-dimensional Gaussian distribution. Here, the zero-variance density is $g^*(\mathbf{x}) = \dfrac{f(\mathbf{x})\mathbb{I}_{\{\varphi(\mathbf{x})\geq 0\}}}{\mathscr{E}}$, and the optimal parameters $\mathbf{m}^*$ and * in Equation 3 can be computed exactly, namely $\mathbf{m}^* = \alpha\mathbf{1}$ with $\alpha = e^{-9/2}/(E(2\pi)^{1/2})$ and $\mathbf{1} = \frac{1}{\sqrt{n}}(1, \dots, 1) \in \mathbb{R}^n$ the normalized constant vector, and $^* = (\nu - 1)\mathbf{1}\mathbf{1}^\top + I_n$ with $\nu = 3\alpha - \alpha^2 + 1$.

5.1.1 Evolution of the partial KL divergence and spectrum

Figure 2a represents the evolution as the dimension varies between 5 and 100 of the partial KL divergence D' for three different choices of covariance matrix: the optimal matrix *, its empirical estimation$\hat{}^*$ and the estimation$\hat{}^*_k$ of the optimal lower-dimensional covariance matrix. We can notice that the partial KL divergence for$\hat{}^*$ grows much faster than the other two, and that the partial KL divergence for$\hat{}^*_k$ remains very close to the optimal value $D'(^*)$. As the KL divergence is a proxy for the efficiency of the auxiliary density (it is for instance closely related to the number of samples required for a given precision (Chatterjee and Diaconis 2018)), this suggests that using$\hat{}^*_k$ will provide results close to optimal.

We now check this claim. As $^* = (\nu - 1)\mathbf{1}\mathbf{1}^\top + I_n$, its eigenpairs are $(\nu, \mathbf{1})$ and $(1, \mathbf{d}_i)$ where the $\mathbf{d}_i$'s form an orthonormal basis of the space orthogonal to the space spanned by $\mathbf{1}$. In particular, $(\nu, \mathbf{1})$ is the largest (in ℓ-order) eigenpair of * and $^*_k = {}^*$ for any $k \geq 1$.

In practice, we do not use this theoretical knowledge and *, $^*_k$ and the eigenpairs are estimated. The six covariance matrices introduced in Section 4.2 and in which we are interested are as follows:

- $^* = (\nu - 1)\mathbf{1}\mathbf{1}^\top + I_n$;
- $\hat{}^*$ given by Equation 9;
- $\hat{}_{\text{opt}}$ and$\hat{}_{\text{mean}}$ are equal and given by $(\hat{\lambda} - 1)\mathbf{1}\mathbf{1}^\top + I_n$ with $\hat{\lambda} = \mathbf{1}^\top\hat{}^*\mathbf{1}$. This amounts to assuming that the projection direction $\mathbf{1}$ is perfectly known, whereas the variance in this direction is estimated;
- $\hat{}^{+\mathrm{d}}_{\text{opt}} = (\hat{\lambda} - 1)\hat{\mathbf{d}}\hat{\mathbf{d}}^\top + I_n$ with $(\hat{\lambda}, \hat{\mathbf{d}})$ the smallest eigenpair of$\hat{}^*$. The difference with the previous case is that we do not assume anymore that the optimal projection direction $\mathbf{1}$ is known, and so it needs to be estimated;
- $\hat{}^{+\mathrm{d}}_{\text{mean}} = (\hat{\lambda} - 1)\frac{\widehat{\mathbf{m}}^*(\widehat{\mathbf{m}}^*)^\top}{\|\widehat{\mathbf{m}}^*\|^2} + I_n$ with $\widehat{\mathbf{m}}^*$ given by Equation 8 and $\hat{\lambda} = \frac{(\widehat{\mathbf{m}}^*)^\top\hat{}^*\widehat{\mathbf{m}}^*}{\|\widehat{\mathbf{m}}^*\|^2}$. Here we assume that $\mathbf{m}^*$ is a good projection direction, but is unknown and therefore needs to be estimated.

Note that in the particularly simple case considered here, both $\widehat{\mathbf{m}}^*/\|\widehat{\mathbf{m}}^*\|$ and $\hat{\mathbf{d}}$ are estimators of $\mathbf{1}$ but they are obtained by different methods. In the next example we will consider a case where $\mathbf{m}^*$ is not an optimal projection direction as given by Theorem 3.1.

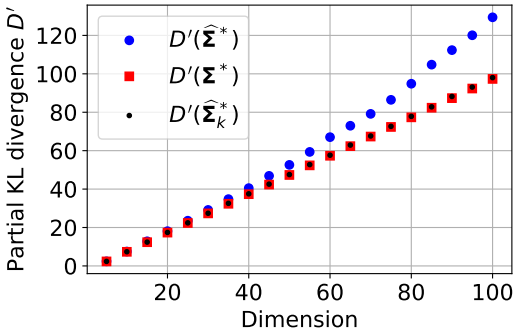
Figure 2b represents the images by ℓ of the eigenvalues of * and$\hat{}^*$. This picture carries a very important insight. We notice that the estimation of most eigenvalues is poor: indeed, all the blue crosses except the leftmost one are meant to be estimator of 1, whereas we see that they are more or less uniformly spread around 1. This means that the variance terms in the corresponding directions are poorly estimated, which could be the explanation on why the use of$\hat{}^*$ gives an inaccurate estimation. But what we remark also is that the function ℓ is quite flat around one: as a consequence, although the eigenvalues offer significant variability, this variability is smoothed by the action of ℓ. Indeed, the

images of the eigenvalues by ℓ take values between 0 and 0.8 and have smaller variability. Moreover, $\ell(x)$ increases sharply as x approaches 0 and thus efficiently distinguishes between the two leftmost estimated eigenvalues and is able to separate them.

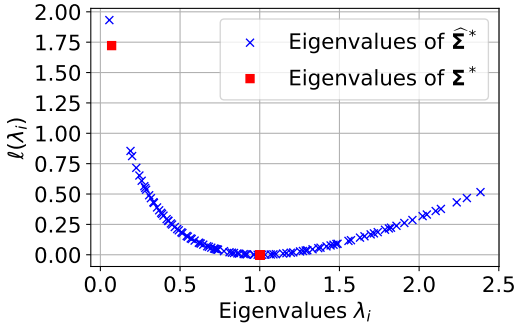
```
################################################################################
# Figure 2. Evolution of the partial KL divergence and spectrum of the
# eigenvalues for the test case 1
################################################################################

def Somme(x):
    n=np.shape(x)[1]
    return(np.sum(x,axis=1)-3*np.sqrt(n))

n=100         # dimension
phi=Somme
E=sp.stats.norm.cdf(-3)   # exact value of the integral

DKL=np.zeros(20)
DKLp=np.zeros(20)
DKLm=np.zeros(20)
DKLstar=np.zeros(20)

M=300

for d in range(5,n+1,5):
    # Mstar
    alpha=np.exp(-3**2/2)/(E*np.sqrt(2*np.pi))
    Mstar=alpha*np.ones(d)/np.sqrt(d)
    # Sigmastar
    vstar=3*alpha-alpha**2+1
    Sigstar= (vstar-1)*np.ones((d,d))/d+np.eye(d)

    ## g*-sample
    VA0=sp.stats.multivariate_normal(mean=np.zeros(d),cov=np.eye(d))
    X0=VA0.rvs(size=M*1000)

    ind=(phi(X0)>0)
    X=X0[ind,:]
    X=X[:M,:]            # g*-sample of size M

    ## estimated mean and covariance
    mm=np.mean(X,axis=0)

    Xc=(X-mm).T
    sigma =Xc @ Xc.T/np.shape(Xc)[1]

    ## projection with the eigenvalues of sigma
    Eig=np.linalg.eigh(sigma)
    logeig=np.sort(np.log(Eig[0])-Eig[0])
```

```
    delta=np.zeros(len(logeig)-1)
    for j in range(len(logeig)-1):
        delta[j]=abs(logeig[j]-logeig[j+1])

    k=np.argmax(delta)+1          # biggest gap between the l(lambda_i)

    indi=[]
    for l in range(k):
        indi.append(np.where(np.log(Eig[0])-Eig[0]==logeig[l])[0][0])

    P1=np.array(Eig[1][:,indi[0]],ndmin=2).T
    for l in range(1,k):
        # matrix of inflential directions of projection
        P1=np.concatenate((P1,np.array(Eig[1][:,indi[l]],ndmin=2).T),axis=1)

    diagsi=np.diag(Eig[0][indi])
    sig_opt_d=P1.dot((diagsi-np.eye(k))).dot(P1.T)+np.eye(d)

    DKL[int((d-5)/5)]=np.log(np.linalg.det(sigma))+np.sum(np.diag(\
                    Sigstar.dot(np.linalg.inv(sigma))))
    DKLp[int((d-5)/5)]=np.log(np.linalg.det(sig_opt_d))+np.sum(\
                    np.diag(Sigstar.dot(np.linalg.inv(sig_opt_d))))
    DKLstar[int((d-5)/5)]=np.log(np.linalg.det(Sigstar))+d

#### plot of partial KL divergence
plt.plot(range(5,105,5),DKL,'bo',label=r"$D'(\widehat{\mathbf{\Sigma}}^*)$")
plt.plot(range(5,105,5),DKLstar,'rs',label=r"$D'(\mathbf{\Sigma}^*)$")
plt.plot(range(5,105,5),DKLp,'k.',label=r"$D'(\widehat{\mathbf{\Sigma}}^*_k)$")

plt.grid()
plt.xlabel('Dimension',fontsize=16)
plt.ylabel(r"Partial KL divergence $D'$",fontsize=16)
plt.legend(fontsize=16)
for tickLabel in plt.gca().get_xticklabels() + plt.gca().get_yticklabels():
    tickLabel.set_fontsize(16)
plt.show()

#### plot of the eigenvalues
Eig1=np.linalg.eigh(sigma)
logeig1=np.log(Eig1[0])-Eig1[0]+1
Table_eigv=np.zeros((n,2))
Table_eigv[:,0]=Eig1[0]
Table_eigv[:,1]=-logeig1

Eigst=np.linalg.eigh(Sigstar)
logeigst=np.log(Eigst[0])-Eigst[0]+1
Table_eigv_st=np.zeros((n,2))
Table_eigv_st[:,0]=Eigst[0]
Table_eigv_st[:,1]=-logeigst
```

```
plt.grid()
plt.xlabel(r"Eigenvalues $\lambda_i$",fontsize=16)
plt.ylabel(r"$\ell(\lambda_i)$",fontsize=16)
for tickLabel in plt.gca().get_xticklabels() + plt.gca().get_yticklabels():
    tickLabel.set_fontsize(16)

plt.plot(Table_eigv[:,0],Table_eigv[:,1],'bx',\
        label=r"Eigenvalues of $\widehat{\mathbf{\Sigma}}^*$")
plt.plot(Table_eigv_st[:,0],Table_eigv_st[:,1],'rs',\
         label=r"Eigenvalues of $\mathbf{\Sigma}^*$")
plt.legend(fontsize=16)
plt.show()
```



(a) Evolution of the partial KL divergence as the dimension increases, with the optimal covariance matrix * (red squares), the sample covariance$\hat{}^*$ (blue circles), and the projected covariance$\hat{}^*_k$ (black dots).



(b) Computation of $\ell(\lambda_i)$ for the eigenvalues of * (red squares) and$\hat{}^*$ (blue crosses) in dimension $n = 100$.

Figure 2: Partial KL divergence and spectrum for the function $\phi = \mathbb{I}_{\varphi \geq 0}$ with φ the linear function given by Equation 10.

5.1.2 Numerical results

We report in Table 2 the numerical results for the six different matrices and the vMFN model for the dimension $n = 100$. The column * gives the optimal results, while the column$\hat{}^*$ corresponds to the results that we are trying to improve. Comparing these two columns, we notice as expected that the estimation of $\mathscr{E}$ with$\hat{}^*$ is significantly degraded. Compared to the first column *, the third and fourth columns with$\hat{}_{\text{opt}} = \hat{}_{\text{mean}}$ correspond to the best projection direction $\mathbf{1}$ (as for *) but estimating the variance in this direction (instead of the true variance) with $\mathbf{1}^\top \hat{}^* \mathbf{1}$. This choice performs very well, with numerical results similar to the optimal ones. This can be understood since in this case, both $\hat{}_{\text{opt}}$ and * are of the form $\alpha \mathbf{1}\mathbf{1}^\top + I_n$ and so estimating$\hat{}_{\text{opt}}$ requires only a one-dimensional estimation (namely, the estimation of α). Next, the last two columns$\hat{}^{+\text{d}}_{\text{opt}}$ and$\hat{}^{+\text{d}}_{\text{mean}}$ highlight the impact of having to estimate the projection directions in addition to the variance since these two matrices are of the form $\hat{\alpha}\hat{\mathbf{1}}\hat{\mathbf{1}}^\top + I_n$ with both $\hat{\alpha}$ (the variance term) and $\hat{\mathbf{1}}$ (the direction) being estimated. We observe that these matrices yield results which are close to optimal and greatly improve the estimation obtained using$\hat{}^*$. In dimension 100, the coefficient of variation is around 5.1% for$\hat{}^{+\text{d}}_{\text{mean}}$, and around 6.2% for $\hat{}^{+\text{d}}_{\text{opt}}$, compared to 2.6% for$\hat{}^*$.

Moreover, we observe that$\hat{}^{+\text{d}}_{\text{mean}}$ gives better results than$\hat{}^{+\text{d}}_{\text{opt}}$. We suggest that this is because $\widehat{\mathbf{m}}^*/\|\widehat{\mathbf{m}}^*\|$

is a better estimator of $\mathbf{1}$ than the eigenvector of $\hat{}^*$. Indeed, evaluating $\widehat{\mathbf{m}}^*$ requires the estimation of n parameters, whereas $\hat{}^*$ needs around $n^2/2$ parameters to estimate, so the eigenvector is finally more noisy than the mean vector. In the last column, we present the vMFN estimation that is slightly more efficicent than the estimation obtained with $\hat{}^{+\mathrm{d}}_{\mathrm{mean}}$.

Thus, the proposed idea improves significantly the probability estimation in high dimensions. But we see that the method taken in (El Masri, Morio, and Simatos 2021) with the projection $\mathbf{m}^*$ is at least as much efficient in this example where we need only a one-dimensional projection. The next case shows that the projection on more than one direction can outperform the one-dimensional projection on $\mathbf{m}^*$.

```
################################################################################
# Table 2. Numerical comparison on test case 1
################################################################################

n=100         # dimension
phi=Somme

def mypi(X):
    n=np.shape(X)[1]
    f0=sp.stats.multivariate_normal.pdf(X,mean=np.zeros(n),cov=np.eye(n))
    return((phi(X)>0)*f0)

N=2000
M=500
B=500   # number of runs

Eopt=np.zeros(B)
EIS=np.zeros(B)
Eprj=np.zeros(B)
Eprm=np.zeros(B)
Eprjst=np.zeros(B)
Eprmst=np.zeros(B)
Evmfn=np.zeros(B)

SI=[]
SIP=[]
SIPst=[]
SIM=[]
SIMst=[]

# Mstar
alpha=np.exp(-3**2/2)/(E*np.sqrt(2*np.pi))
Mstar=alpha*np.ones(d)/np.sqrt(d)
# Sigmastar
vstar=3*alpha-alpha**2+1
Sigstar= (vstar-1)*np.ones((d,d))/d+np.eye(d)

Eigst=np.linalg.eigh(Sigstar)
logeigst=np.sort(np.log(Eigst[0])-Eigst[0])
```

```
deltast=np.zeros(len(logeigst)-1)

for i in range(len(logeigst)-1):
    deltast[i]=abs(logeigst[i]-logeigst[i+1])

## choice of the number of dimension
k_st=np.argmax(deltast)+1

indist=[]
for i in range(k_st):
    indist.append(np.where(np.log(Eigst[0])-Eigst[0]==logeigst[i])[0][0])

P1st=np.array(Eigst[1][:,indist[0]],ndmin=2).T
for i in range(1,k_st):
    P1st=np.concatenate((P1st,np.array(Eigst[1][:,indist[i]],ndmin=2).T)\
                        ,axis=1)    # matrix of influential directions

#np.random.seed(0)
for i in range(B):
############################## Estimation of the matrices

   ## g*-sample of size M
    VA=sp.stats.multivariate_normal(np.zeros(n),np.eye(n))
    X0=VA.rvs(size=M*1000)
    ind=(phi(X0)>0)
    X1=X0[ind,:]
    X=X1[:M,:]

    R=np.sqrt(np.sum(X**2,axis=1))
    Xu=(X.T/R).T

   ## estimated gaussian mean and covariance
    mm=np.mean(X,axis=0)
    Xc=(X-mm).T
    sigma =Xc @ Xc.T/np.shape(Xc)[1]
    SI.append(sigma)

   ## von Mises Fisher parameters
    normu=np.sqrt(np.mean(Xu,axis=0).dot(np.mean(Xu,axis=0).T))
    mu=np.mean(Xu,axis=0)/normu
    mu=np.array(mu,ndmin=2)
    chi=min(normu,0.95)
    kappa=(chi*n-chi**3)/(1-chi**2)

   ## Nakagami parameters
    omega=np.mean(R**2)
    tau4=np.mean(R**4)
    pp=omega**2/(tau4-omega**2)
```

```
    ###
     Eig=np.linalg.eigh(sigma)
     logeig=np.sort(np.log(Eig[0])-Eig[0])
     delta=np.zeros(len(logeig)-1)
     for j in range(len(logeig)-1):
         delta[j]=abs(logeig[j]-logeig[j+1])

     k=np.argmax(delta)+1

     indi=[]
     for l in range(k):
         indi.append(np.where(np.log(Eig[0])-Eig[0]==logeig[l])[0][0])

     P1=np.array(Eig[1][:,indi[0]],ndmin=2).T
     for l in range(1,k):
         P1=np.concatenate((P1,np.array(Eig[1][:,indi[l]],ndmin=2).T)\
                           ,axis=1)

     diagsi=np.diag(Eig[0][indi])
     sig_opt_d=P1.dot((diagsi-np.eye(k))).dot(P1.T)+np.eye(n)
     SIP.append(sig_opt_d)

    ###
     diagsist=P1st.T.dot(sigma).dot(P1st)
     sig_opt=P1st.dot(diagsist-np.eye(k_st)).dot(P1st.T)+np.eye(n)
     SIPst.append(sig_opt)

    ###
     Norm_mm=np.linalg.norm(mm)
     normalised_mm=np.array(mm,ndmin=2).T/Norm_mm
     vhat=normalised_mm.T.dot(sigma).dot(normalised_mm)
     sig_mean_d=(vhat-1)*normalised_mm.dot(normalised_mm.T)+np.eye(n)
     SIM.append(sig_mean_d)

############################################## Estimation of the integral
    ###
     Xop=sp.stats.multivariate_normal.rvs(mean=mm, cov=Sigstar,size=N)
     wop=mypi(Xop)/sp.stats.multivariate_normal.pdf(Xop,mean=mm, cov=Sigstar)
     Eopt[i]=np.mean(wop)

    ###
     Xis=sp.stats.multivariate_normal.rvs(mean=mm, cov=sigma,size=N)
     wis=mypi(Xis)/sp.stats.multivariate_normal.pdf(Xis,mean=mm, cov=sigma)
     EIS[i]=np.mean(wis)

    ###
     Xpr=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt_d,size=N)
     wpr=mypi(Xpr)/sp.stats.multivariate_normal.pdf(Xpr,mean=mm,\
                                                    cov=sig_opt_d)
```

```
    Eprj[i]=np.mean(wpr)

   ###
    Xpm=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_mean_d,size=N)
    wpm=mypi(Xpm)/sp.stats.multivariate_normal.pdf(Xpm,mean=mm,\
                                                   cov=sig_mean_d)
    Eprm[i]=np.mean(wpm)

   ###
    Xprst=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt,size=N)
    wprst=mypi(Xprst)/sp.stats.multivariate_normal.pdf(Xprst,mean=mm, \
                                                       cov=sig_opt)
    Eprjst[i]=np.mean(wprst)

   ###
    Xvmfn = vMFNM_sample(mu, kappa, omega, pp, 1, N)
    Rvn=np.sqrt(np.sum(Xvmfn**2,axis=1))
    Xvnu=Xvmfn.T/Rvn

    h_log=vMF_logpdf(Xvnu,mu.T,kappa)+nakagami_logpdf(Rvn,pp,omega)
    A = np.log(n) + np.log(np.pi ** (n / 2)) - sp.special.gammaln(n / 2 + 1)
    f_u = -A
    f_chi = (np.log(2) * (1 - n / 2) + np.log(Rvn) * (n - 1) - 0.5 * \
             Rvn ** 2 - sp.special.gammaln(n / 2))
    f_log = f_u + f_chi
    W_log = f_log - h_log

    wvmfn=(phi(Xvmfn)>0)*np.exp(W_log)
    Evmfn[i]=np.mean(wvmfn)

### KL divergences
dkli=np.zeros(B)
dklp=np.zeros(B)
dklm=np.zeros(B)
dklpst=np.zeros(B)

for i in range(B):
    dkli[i]=np.log(np.linalg.det(SI[i]))+sum(np.diag(Sigstar.dot\
                                            (np.linalg.inv(SI[i]))))
    dklp[i]=np.log(np.linalg.det(SIP[i]))+sum(np.diag(Sigstar.dot\
                                            (np.linalg.inv(SIP[i]))))
    dklm[i]=np.log(np.linalg.det(SIM[i]))+sum(np.diag(Sigstar.dot\
                                            (np.linalg.inv(SIM[i]))))
    dklpst[i]=np.log(np.linalg.det(SIPst[i]))+sum(np.diag(Sigstar.dot\
                                            (np.linalg.inv(SIPst[i]))))

Tabresult=np.zeros((3,7)) # table of results

Tabresult[0,0]=np.log(np.linalg.det(Sigstar))+n
```

```
Tabresult[0,1]=np.mean(dkli)
Tabresult[0,2]=np.mean(dklpst)
Tabresult[0,3]=np.mean(dklpst)
Tabresult[0,4]=np.mean(dklp)
Tabresult[0,5]=np.mean(dklm)
Tabresult[0,6]=None

Tabresult[1,0]=np.mean(Eopt-E)/E*100
Tabresult[1,1]=np.mean(EIS-E)/E*100
Tabresult[1,2]=np.mean(Eprjst-E)/E*100
Tabresult[1,3]=np.mean(Eprjst-E)/E*100
Tabresult[1,4]=np.mean(Eprj-E)/E*100
Tabresult[1,5]=np.mean(Eprm-E)/E*100
Tabresult[1,6]=np.mean(Evmfn-E)/E*100

Tabresult[2,0]=np.sqrt(np.mean((Eopt-E)**2))/E*100
Tabresult[2,1]=np.sqrt(np.mean((EIS-E)**2))/E*100
Tabresult[2,2]=np.sqrt(np.mean((Eprjst-E)**2))/E*100
Tabresult[2,3]=np.sqrt(np.mean((Eprjst-E)**2))/E*100
Tabresult[2,4]=np.sqrt(np.mean((Eprj-E)**2))/E*100
Tabresult[2,5]=np.sqrt(np.mean((Eprm-E)**2))/E*100
Tabresult[2,6]=np.sqrt(np.mean((Evmfn-E)**2))/E*100

Tabresult=np.round(Tabresult,1)

table=[["D'",Tabresult[0,0],Tabresult[0,1],Tabresult[0,2],Tabresult[0,3],
        Tabresult[0,4],Tabresult[0,5],"/"],
       ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],
        Tabresult[1,2],Tabresult[1,3],Tabresult[1,4],Tabresult[1,5],Tabresult[1,6]],
    ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],
     Tabresult[2,2],Tabresult[2,3],Tabresult[2,4],Tabresult[2,5],Tabresult[2,6]]]

Markdown(tabulate(
  table,
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}
           "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
           "$\widehat{\mathbf{\Sigma}}^{+d}_{mean}$", "vMFN"],
    tablefmt="pipe"))
```

```
<>:214: SyntaxWarning: invalid escape sequence '\%'
<>:216: SyntaxWarning: invalid escape sequence '\%'
<>:221: SyntaxWarning: invalid escape sequence '\m'
<>:221: SyntaxWarning: invalid escape sequence '\w'
<>:221: SyntaxWarning: invalid escape sequence '\w'
<>:222: SyntaxWarning: invalid escape sequence '\w'
<>:222: SyntaxWarning: invalid escape sequence '\w'
<>:223: SyntaxWarning: invalid escape sequence '\w'
<>:214: SyntaxWarning: invalid escape sequence '\%'
<>:216: SyntaxWarning: invalid escape sequence '\%'
<>:221: SyntaxWarning: invalid escape sequence '\m'
```

```
<>:221: SyntaxWarning: invalid escape sequence '\w'
<>:221: SyntaxWarning: invalid escape sequence '\w'
<>:222: SyntaxWarning: invalid escape sequence '\w'
<>:222: SyntaxWarning: invalid escape sequence '\w'
<>:223: SyntaxWarning: invalid escape sequence '\w'
/tmp/ipykernel_7763/4266669899.py:214: SyntaxWarning: invalid escape sequence '\%'
  ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],
/tmp/ipykernel_7763/4266669899.py:216: SyntaxWarning: invalid escape sequence '\%'
  ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],
/tmp/ipykernel_7763/4266669899.py:221: SyntaxWarning: invalid escape sequence '\m'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/4266669899.py:221: SyntaxWarning: invalid escape sequence '\w'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/4266669899.py:221: SyntaxWarning: invalid escape sequence '\w'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/4266669899.py:222: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
/tmp/ipykernel_7763/4266669899.py:222: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
/tmp/ipykernel_7763/4266669899.py:223: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}^{+d}_{mean}$", "vMFN"],
```

Table 2: Numerical comparison of the estimation of $\mathscr{E} \approx 1.35 \cdot 10^{-3}$ considering the Gaussian model with the six covariance matrices defined in Section 4.2 and the vFMN model, when $\phi = \mathbb{I}_{\varphi \geq 0}$ with φ the linear function given by Equation 10. As explained in the text, $\hat{}_{\text{mean}}$ and $\hat{}_{\text{opt}}$ are actually equal in this case. The computational cost is $N = 2000$.

	$*$	$\hat{}*$	$\hat{}_{opt}$	$\hat{}_{mean}$	$\hat{}^{+d}_{opt}$	$\hat{}^{+d}_{mean}$	vMFN
D'	97.3	111.9	97.4	97.4	97.7	97.5	/
Relative error (%)	-0.1	-24.3	-0.1	-0.1	-0.3	0.2	0.2
Coefficient of variation (%)	2.6	149.1	3.7	3.7	6.2	5.1	4.5

5.2 Test case 2: projection in 2 directions

The second test case is again a probability estimation, i.e., it is of the form $\phi = \mathbb{I}_{\{\varphi \geq 0\}}$ with now the function φ having some quadratic terms:

$$\varphi : \mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n \mapsto x_1 - 25x_2^2 - 30x_3^2 - 1. \tag{11}$$

The quantity of interest $\mathscr{E}$ is defined as $\mathscr{E} = \int_{\mathbb{R}^n} \phi(\mathbf{x}) f(\mathbf{x}) \mathrm{d}\mathbf{x} = \mathbb{P}_f(\varphi(\mathbf{X}) \geq 0)$ for all n where the density f is the standard n-dimensional Gaussian distribution. This function is motivated in part because $\mathbf{m}^*$ and $\mathbf{d}_1^*$ are different and also because Algorithm 2 chooses two projection directions. Thus, this is an example where $\hat{}_{\text{mean}}$ and $\hat{}_{\text{opt}}$ are significantly different.

5.2.1 Evolution of the partial KL divergence and spectrum

We check on Figure 3a that the partial KL divergence obeys the same behavior as for the previous example, namely the one associated with $\hat{}^*$ increases much faster than the ones associated with * and $\hat{}^*_k$, which again suggests that projecting can improve the situation. Since the function φ only depends on the first three variables and is even in x_2 and x_3, one gets that $\mathbf{m}^* = \alpha \mathbf{e}_1$ with $\alpha = \mathbb{E}(X_1 \mid X_1 \geq 25X_2^2 + 30X_3^2 + 1) \approx 1.9$ (here and in the sequel, $\mathbf{e}_i$ denotes the ith canonical vector of $\mathbb{R}^n$, i.e., all its coordinates are 0 except the i-th one which is equal to one), and that * is diagonal with

$$^* = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \lambda_3 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

Note that the off-diagonal elements of the submatrix $(^*_{ij})_{1\leq i,j\leq 3}$ are indeed 0 since they result from integrating an odd function of an odd random variable with an even conditioning. For instance, if $F(x) = \mathbb{P}(30X_3^2 + 1 \leq x)$, then by conditioning on (X_1, X_3) we obtain

$$^*_{12} = \mathbb{E}\left((X_1 - \alpha)X_2 \mid X_1 - 25X_2^2 \geq 30X_3^2 + 1\right) = \frac{1}{\mathscr{C}}\mathbb{E}\left[(X_1 - \alpha)\mathbb{E}\left(X_2 F(X_1 - 25X_2^2) \mid X_1\right)\right]$$

which is 0 as $x_2 F(x_1 - x_2^2)$ is an odd function of x_2 for fixed x_1, and X_2 has an even density.

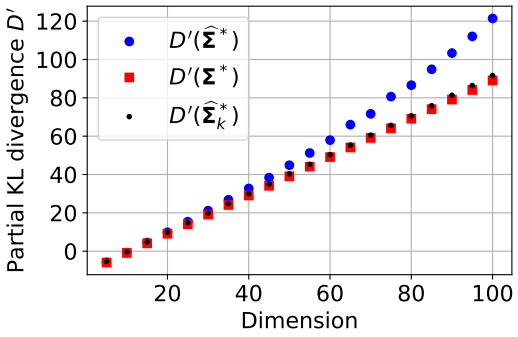
We can numerically compute $\lambda_1 \approx 0.28$, $\lambda_2 \approx 0.009$ and $\lambda_3 \approx 0.008$. These values correspond to the red squares in Figure 3b which shows that the smallest eigenvalues are properly estimated. Moreover, Algorithm 2 selects the two largest eigenvalues, which have the highest ℓ-values. These two eigenvalues thus correspond to the eigenvectors $\mathbf{e}_2$ and $\mathbf{e}_3$, and so we see that on this example, the optimal directions predicted by Theorem 3.1 are significantly different (actually, orthogonal) from $\mathbf{m}^*$ which is proportional to $\mathbf{e}_1$.

```
##############################################################################
# Figure 3. Evolution of the partial KL divergence and spectrum of the
# eigenvalues for the test case 2
##############################################################################
#E=1.51*10**-3
def parabol(X):
    n=np.shape(X)[1]
    return(X[:,0]-25*X[:,1]**2-30*X[:,2]**2-1)

bigsample=1*10**8
phi=parabol

VA0=sp.stats.multivariate_normal(mean=np.zeros(3),cov=np.eye(3))
X0=VA0.rvs(size=bigsample)

ind=(phi(X0)>0)
X=X0[ind,:]

E=1.51*10**-3   # reference value of the integral

Mstar_dim3=np.zeros(3)
```

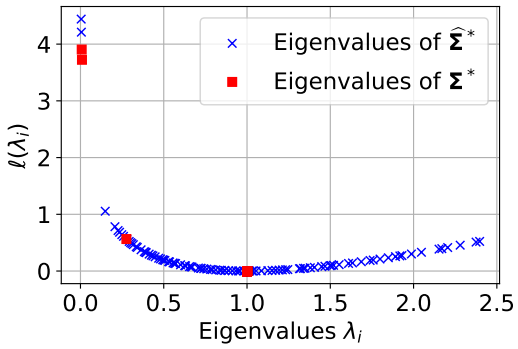
```
# accurate value of optimal mean in dimension 3
Mstar_dim3[0]=np.mean(X,axis=0)[0]
Xc=(X-Mstar_dim3).T
# accurate value of optimal covariance in dimension 3
Sigstar_dim3=Xc @ Xc.T/np.shape(Xc)[1]

DKL=np.zeros(20)
DKLp=np.zeros(20)
DKLm=np.zeros(20)
DKLstar=np.zeros(20)

M=300

for d in range(5,n+1,5):

    # Mstar
    Mstar=np.zeros(d)
    Mstar[:3]=Mstar_dim3
    # Sigmastar
    Sigstar=np.eye(d)
    Sigstar[:3,:3]=Sigstar_dim3

    ## g*-sample
    VA0=sp.stats.multivariate_normal(mean=np.zeros(d),cov=np.eye(d))
    X0=VA0.rvs(size=M*1000)

    ind=(phi(X0)>0)
    X=X0[ind,:]
    X=X[:M,:]

    ## estimated mean and covariance
    mm=np.mean(X,axis=0)

    Xc=(X-mm).T
    sigma =Xc @ Xc.T/np.shape(Xc)[1]

    ## projection with the eigenvalues of sigma
    Eig=np.linalg.eigh(sigma)
    logeig=np.sort(np.log(Eig[0])-Eig[0])
    delta=np.zeros(len(logeig)-1)
    for j in range(len(logeig)-1):
        delta[j]=abs(logeig[j]-logeig[j+1])

    k=np.argmax(delta)+1          # biggest gap between the l(lambda_i)

    indi=[]
    for l in range(k):
        indi.append(np.where(np.log(Eig[0])-Eig[0]==logeig[l])[0][0])
```

```
    P1=np.array(Eig[1][:,indi[0]],ndmin=2).T
    for l in range(1,k):
        # matrix od influential directions of projections
        P1=np.concatenate((P1,np.array(Eig[1][:,indi[l]],ndmin=2).T),axis=1)

    diagsi=np.diag(Eig[0][indi])
    sig_opt_d=P1.dot((diagsi-np.eye(k))).dot(P1.T)+np.eye(d)

    DKL[int((d-5)/5)]=np.log(np.linalg.det(sigma))+np.sum(np.diag(Sigstar.dot\
                                                    (np.linalg.inv(sigma))))
    DKLp[int((d-5)/5)]=np.log(np.linalg.det(sig_opt_d))+np.sum(np.diag(\
                                    Sigstar.dot(np.linalg.inv(sig_opt_d))))
    DKLstar[int((d-5)/5)]=np.log(np.linalg.det(Sigstar))+d

#### plot of partial KL divergence
plt.plot(range(5,105,5),DKL,'bo',label=r"$D'(\widehat{\mathbf{\Sigma}}^*)$")
plt.plot(range(5,105,5),DKLstar,'rs',label=r"$D'(\mathbf{\Sigma}^*)$")
plt.plot(range(5,105,5),DKLp,'k.',label=r"$D'(\widehat{\mathbf{\Sigma}}^*_k)$")

plt.grid()
plt.xlabel('Dimension',fontsize=16)
plt.ylabel(r"Partial KL divergence $D'$",fontsize=16)
plt.legend(fontsize=16)
for tickLabel in plt.gca().get_xticklabels() + plt.gca().get_yticklabels():
    tickLabel.set_fontsize(16)
plt.show()

#### plot of the eigenvalues
Eig1=np.linalg.eigh(sigma)
logeig1=np.log(Eig1[0])-Eig1[0]+1
Table_eigv=np.zeros((n,2))
Table_eigv[:,0]=Eig1[0]
Table_eigv[:,1]=-logeig1

Eigst=np.linalg.eigh(Sigstar)
logeigst=np.log(Eigst[0])-Eigst[0]+1
Table_eigv_st=np.zeros((n,2))
Table_eigv_st[:,0]=Eigst[0]
Table_eigv_st[:,1]=-logeigst

plt.grid()
plt.xlabel(r"Eigenvalues $\lambda_i$",fontsize=16)
plt.ylabel(r"$\ell(\lambda_i)$",fontsize=16)
for tickLabel in plt.gca().get_xticklabels() + plt.gca().get_yticklabels():
    tickLabel.set_fontsize(16)

plt.plot(Table_eigv[:,0],Table_eigv[:,1],'bx',\
         label=r"Eigenvalues of $\widehat{\mathbf{\Sigma}}^*$")
plt.plot(Table_eigv_st[:,0],Table_eigv_st[:,1],'rs',\
```

```
        label=r"Eigenvalues of $\mathbf{\Sigma}^*$")
plt.legend(fontsize=16)
plt.show()
```



(a) Evolution of the partial KL divergence as the dimension increases, with the optimal covariance matrix * (red squares), the sample covariance$\hat{}^*$ (blue circles), and the projected covariance$\hat{}^*_k$ (black dots).



(b) Computation of $\ell(\lambda_i)$ for the eigenvalues of * (red squares) and$\hat{}^*$ (blue crosses) in dimension $n = 100$.

Figure 3: Partial KL divergence and spectrum for the function $\phi = \mathbb{I}_{\varphi \geq 0}$ with φ given by Equation 11. in dimension $n = 100$. Left: same behavior as for the first test case. Right: we now have two eigenvalues that stand out, and the behavior of ℓ is such that Algorithm 2 selects $k = 2$ which corresponds to the leftmost two.

5.2.2 Numerical results

The numerical results of our simulations are presented in Table 3. We remark as before that, when using$\hat{}^*$, the accuracy quickly deteriorates as the dimension increases as shows the coefficient of variation of 396% in dimension $n = 100$. In contrast,$\hat{}_{\text{opt}}$ leads to very accurate results, which remain close to optimal up to the same dimension $n = 100$. This behavior is to compare with the evolution of the relative KL divergence: contrary to$\hat{}^*$,$\hat{}_{\text{opt}}$ gives a partial KL divergence close to optimal in dimension $n = 100$. This confirms that the KL divergence is indeed a good proxy to assess the relevance of an auxiliary density.

It is also interesting to note that the direction $\mathbf{m}^*$ improves the situation compared to not projecting (column$\hat{}_{\text{mean}}$ compared to$\hat{}^*$), but using$\hat{}_{\text{opt}}$ gives significantly better results, with for instance a coefficient of variation around 3.6% for$\hat{}_{\text{opt}}$ and around 28.8% for$\hat{}_{\text{mean}}$ in dimension $n = 100$. Thus, this confirms our theoretical result that the $\mathbf{d}_i^*$'s are good directions on which to project.

Finally, we notice that performing estimations of the projection directions instead of taking the true ones (columns$\hat{}^{+\text{d}}_{\text{opt}}$ vs$\hat{}_{\text{opt}}$) slightly degrades the situation, making the coefficient of variation increase from 3.6 to 9.5% even if the accuracy remains satisfactory. The vMFN model is also not really adapted to this example as it gives results similar to$\hat{}_{\text{mean}}$. Gaussian density family are more able to fit g^* than vMFN parametric model in this test case.

```
################################################################################
# Table 3. Numerical comparison on test case 2
################################################################################

n=100          # dimension
```

```
phi=parabol

def mypi(X):
    n=np.shape(X)[1]
    f0=sp.stats.multivariate_normal.pdf(X,mean=np.zeros(n),cov=np.eye(n))
    return((phi(X)>0)*f0)

N=2000
M=500
B=500    # number of runs

Eopt=np.zeros(B)
EIS=np.zeros(B)
Eprj=np.zeros(B)
Eprm=np.zeros(B)
Eprjst=np.zeros(B)
Eprmst=np.zeros(B)
Evmfn=np.zeros(B)

SI=[]
SIP=[]
SIPst=[]
SIM=[]
SIMst=[]

bigsample=1*10**8

VA0=sp.stats.multivariate_normal(mean=np.zeros(3),cov=np.eye(3))
X0=VA0.rvs(size=bigsample)

ind=(phi(X0)>0)
X=X0[ind,:]

E=1.51*10**-3   # reference value of the integral

Mstar_dim3=np.zeros(3)
 # accurate value of optimal mean in dimension 3
Mstar_dim3[0]=np.mean(X,axis=0)[0]
Xc=(X-Mstar_dim3).T
# accurate value of optimal covariance in dimension 3
Sigstar_dim3=Xc @ Xc.T/np.shape(Xc)[1]

# Mstar
Mstar=np.zeros(n)
Mstar[:3]=Mstar_dim3
# Sigmastar
Sigstar=np.eye(n)
Sigstar[:3,:3]=Sigstar_dim3
```

```
Eigst=np.linalg.eigh(Sigstar)
logeigst=np.sort(np.log(Eigst[0])-Eigst[0])
deltast=np.zeros(len(logeigst)-1)

for i in range(len(logeigst)-1):
    deltast[i]=abs(logeigst[i]-logeigst[i+1])

## choice of the number of dimension
k_st=np.argmax(deltast)+1

indist=[]
for i in range(k_st):
    indist.append(np.where(np.log(Eigst[0])-Eigst[0]==logeigst[i])[0][0])

P1st=np.array(Eigst[1][:,indist[0]],ndmin=2).T
for i in range(1,k_st):
    # matrix of influential directions
    P1st=np.concatenate((P1st,np.array(Eigst[1][:,indist[i]],ndmin=2).T),axis=1)

for i in range(B):
############################## Estimation of the matrices

   ## g*-sample of size M
    VA=sp.stats.multivariate_normal(np.zeros(n),np.eye(n))
    X0=VA.rvs(size=M*1000)
    ind=(phi(X0)>0)
    X1=X0[ind,:]
    X=X1[:M,:]

    R=np.sqrt(np.sum(X**2,axis=1))
    Xu=(X.T/R).T

   ## estimated gaussian mean and covariance
    mm=np.mean(X,axis=0)
    Xc=(X-mm).T
    sigma =Xc @ Xc.T/np.shape(Xc)[1]
    SI.append(sigma)

   ## von Mises Fisher parameters
    normu=np.sqrt(np.mean(Xu,axis=0).dot(np.mean(Xu,axis=0).T))
    mu=np.mean(Xu,axis=0)/normu
    mu=np.array(mu,ndmin=2)
    chi=min(normu,0.95)
    kappa=(chi*n-chi**3)/(1-chi**2)

   ## Nakagami parameters
    omega=np.mean(R**2)
    tau4=np.mean(R**4)
    pp=omega**2/(tau4-omega**2)
```

```
    ###
     Eig=np.linalg.eigh(sigma)
     logeig=np.sort(np.log(Eig[0])-Eig[0])
     delta=np.zeros(len(logeig)-1)
     for j in range(len(logeig)-1):
         delta[j]=abs(logeig[j]-logeig[j+1])

     k=np.argmax(delta)+1

     indi=[]
     for l in range(k):
         indi.append(np.where(np.log(Eig[0])-Eig[0]==logeig[l])[0][0])

     P1=np.array(Eig[1][:,indi[0]],ndmin=2).T
     for l in range(1,k):
         P1=np.concatenate((P1,np.array(Eig[1][:,indi[l]],ndmin=2).T),axis=1)

     diagsi=np.diag(Eig[0][indi])
     sig_opt_d=P1.dot((diagsi-np.eye(k))).dot(P1.T)+np.eye(n)
     SIP.append(sig_opt_d)

    ###
     diagsist=P1st.T.dot(sigma).dot(P1st)
     sig_opt=P1st.dot(diagsist-np.eye(k_st)).dot(P1st.T)+np.eye(n)
     SIPst.append(sig_opt)

    ###
     Norm_mm=np.linalg.norm(mm)
     normalised_mm=np.array(mm,ndmin=2).T/Norm_mm
     vhat=normalised_mm.T.dot(sigma).dot(normalised_mm)
     sig_mean_d=(vhat-1)*normalised_mm.dot(normalised_mm.T)+np.eye(n)
     SIM.append(sig_mean_d)

    ###
     Norm_Mstar=np.linalg.norm(Mstar)
     normalised_Mstar=np.array(Mstar,ndmin=2).T/Norm_Mstar
     vhatst=normalised_Mstar.T.dot(sigma).dot(normalised_Mstar)

     sig_mean=(vhatst-1)*normalised_Mstar.dot(normalised_Mstar.T)+np.eye(n)
     SIMst.append(sig_mean)

############################################## Estimation of the integral
    ###
     Xop=sp.stats.multivariate_normal.rvs(mean=mm, cov=Sigstar,size=N)
     wop=mypi(Xop)/sp.stats.multivariate_normal.pdf(Xop,mean=mm, cov=Sigstar)
     Eopt[i]=np.mean(wop)

    ###
```

```
    Xis=sp.stats.multivariate_normal.rvs(mean=mm, cov=sigma,size=N)
    wis=mypi(Xis)/sp.stats.multivariate_normal.pdf(Xis,mean=mm, cov=sigma)
    EIS[i]=np.mean(wis)

   ###
    Xpr=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt_d,size=N)
    wpr=mypi(Xpr)/sp.stats.multivariate_normal.pdf(Xpr,mean=mm, cov=sig_opt_d)
    Eprj[i]=np.mean(wpr)

   ###
    Xpm=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_mean_d,size=N)
    wpm=mypi(Xpm)/sp.stats.multivariate_normal.pdf(Xpm,mean=mm, cov=sig_mean_d)
    Eprm[i]=np.mean(wpm)

   ###
    Xprst=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt,size=N)
    wprst=mypi(Xprst)/sp.stats.multivariate_normal.pdf(Xprst,mean=mm, \
                                                       cov=sig_opt)
    Eprjst[i]=np.mean(wprst)

   ###
    Xpmst=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_mean,size=N)
    wpmst=mypi(Xpmst)/sp.stats.multivariate_normal.pdf(Xpmst,mean=mm, \
                                                       cov=sig_mean)
    Eprmst[i]=np.mean(wpmst)

   ###
    Xvmfn = vMFNM_sample(mu, kappa, omega, pp, 1, N)
    Rvn=np.sqrt(np.sum(Xvmfn**2,axis=1))
    Xvnu=Xvmfn.T/Rvn

    h_log=vMF_logpdf(Xvnu,mu.T,kappa)+nakagami_logpdf(Rvn,pp,omega)
    A = np.log(n) + np.log(np.pi ** (n / 2)) - sp.special.gammaln(n / 2 + 1)
    f_u = -A
    f_chi = (np.log(2) * (1 - n / 2) + np.log(Rvn) * (n - 1) - 0.5\
             * Rvn ** 2 - sp.special.gammaln(n / 2))
    f_log = f_u + f_chi
    W_log = f_log - h_log

    wvmfn=(phi(Xvmfn)>0)*np.exp(W_log)
    Evmfn[i]=np.mean(wvmfn)

### KL divergences
dkli=np.zeros(B)
dklp=np.zeros(B)
dklm=np.zeros(B)
dklpst=np.zeros(B)
dklmst=np.zeros(B)
```

```
for i in range(B):
    dkli[i]=np.log(np.linalg.det(SI[i]))+sum(np.diag(Sigstar\
                                            .dot(np.linalg.inv(SI[i]))))
    dklp[i]=np.log(np.linalg.det(SIP[i]))+sum(np.diag(Sigstar\
                                            .dot(np.linalg.inv(SIP[i]))))
    dklm[i]=np.log(np.linalg.det(SIM[i]))+sum(np.diag(Sigstar\
                                            .dot(np.linalg.inv(SIM[i]))))
    dklpst[i]=np.log(np.linalg.det(SIPst[i]))+sum(np.diag(Sigstar\
                                            .dot(np.linalg.inv(SIPst[i]))))
    dklmst[i]=np.log(np.linalg.det(SIMst[i]))+sum(np.diag(Sigstar\
                                            .dot(np.linalg.inv(SIMst[i]))))

Tabresult=np.zeros((3,7)) # table of results

Tabresult[0,0]=np.log(np.linalg.det(Sigstar))+n
Tabresult[0,1]=np.mean(dkli)
Tabresult[0,2]=np.mean(dklpst)
Tabresult[0,3]=np.mean(dklmst)
Tabresult[0,4]=np.mean(dklp)
Tabresult[0,5]=np.mean(dklm)
Tabresult[0,6]=None

Tabresult[1,0]=np.mean(Eopt-E)/E*100
Tabresult[1,1]=np.mean(EIS-E)/E*100
Tabresult[1,2]=np.mean(Eprjst-E)/E*100
Tabresult[1,3]=np.mean(Eprmst-E)/E*100
Tabresult[1,4]=np.mean(Eprj-E)/E*100
Tabresult[1,5]=np.mean(Eprm-E)/E*100
Tabresult[1,6]=np.mean(Evmfn-E)/E*100

Tabresult[2,0]=np.sqrt(np.mean((Eopt-E)**2))/E*100
Tabresult[2,1]=np.sqrt(np.mean((EIS-E)**2))/E*100
Tabresult[2,2]=np.sqrt(np.mean((Eprjst-E)**2))/E*100
Tabresult[2,3]=np.sqrt(np.mean((Eprmst-E)**2))/E*100
Tabresult[2,4]=np.sqrt(np.mean((Eprj-E)**2))/E*100
Tabresult[2,5]=np.sqrt(np.mean((Eprm-E)**2))/E*100
Tabresult[2,6]=np.sqrt(np.mean((Evmfn-E)**2))/E*100

Tabresult=np.round(Tabresult,1)

table=[["D'",Tabresult[0,0],Tabresult[0,1],Tabresult[0,2],Tabresult[0,3],
        Tabresult[0,4],Tabresult[0,5],"/"],
      ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],
       Tabresult[1,2],Tabresult[1,3],Tabresult[1,4],Tabresult[1,5],Tabresult[1,6]],
    ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],
     Tabresult[2,2],Tabresult[2,3],Tabresult[2,4],Tabresult[2,5],Tabresult[2,6]]]
Markdown(tabulate(
  table,
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}
```

```
            "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
            "$\widehat{\mathbf{\Sigma}}^{+d}_{mean}$", "vMFN"],
      tablefmt="pipe"))
```

```
<>:244: SyntaxWarning: invalid escape sequence '\%'
<>:246: SyntaxWarning: invalid escape sequence '\%'
<>:250: SyntaxWarning: invalid escape sequence '\m'
<>:250: SyntaxWarning: invalid escape sequence '\w'
<>:250: SyntaxWarning: invalid escape sequence '\w'
<>:251: SyntaxWarning: invalid escape sequence '\w'
<>:251: SyntaxWarning: invalid escape sequence '\w'
<>:252: SyntaxWarning: invalid escape sequence '\w'
<>:244: SyntaxWarning: invalid escape sequence '\%'
<>:246: SyntaxWarning: invalid escape sequence '\%'
<>:250: SyntaxWarning: invalid escape sequence '\m'
<>:250: SyntaxWarning: invalid escape sequence '\w'
<>:250: SyntaxWarning: invalid escape sequence '\w'
<>:251: SyntaxWarning: invalid escape sequence '\w'
<>:251: SyntaxWarning: invalid escape sequence '\w'
<>:252: SyntaxWarning: invalid escape sequence '\w'
/tmp/ipykernel_7763/1713126179.py:244: SyntaxWarning: invalid escape sequence '\%'
  ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],
/tmp/ipykernel_7763/1713126179.py:246: SyntaxWarning: invalid escape sequence '\%'
  ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],
/tmp/ipykernel_7763/1713126179.py:250: SyntaxWarning: invalid escape sequence '\m'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/1713126179.py:250: SyntaxWarning: invalid escape sequence '\w'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/1713126179.py:250: SyntaxWarning: invalid escape sequence '\w'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/1713126179.py:251: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
/tmp/ipykernel_7763/1713126179.py:251: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
/tmp/ipykernel_7763/1713126179.py:252: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}^{+d}_{mean}$", "vMFN"],
```

Table 3: Numerical comparison of the estimation of $\mathscr{E} \approx 1.51 \cdot 10^{-3}$ considering the Gaussian density with the six covariance matrices defined in Section 4.2 and the vFMN model, when $\phi = \mathbb{I}_{\varphi \geq 0}$ with φ the quadratic function given by Equation 11. The computational cost is $N = 2000$.

	$*$	$\hat{}*$	$\hat{}_{opt}$	$\hat{}_{mean}$	$\hat{}^{+d}_{opt}$	$\hat{}^{+d}_{mean}$	vMFN
D’	89.1	103.6	89.7	96.7	90.4	96.8	/
Relative error (%)	-0.2	9	-0.4	2	-0.3	0.4	1.7
Coefficient of variation (%)	3.5	396.2	3.6	28.8	9.5	32.9	31.9

Remark. For the two test cases studied so far, projecting$\hat{}^*$ in the Failure-Informed Subspace (FIS) of (Uribe et al. 2021) (see the introduction) would outperform our method with$\hat{}^*_k$, leading to results close to those obtained with *. However, computing the FIS relies on the knowledge of the gradient of the function φ, which is straightforward to compute in these two test cases, and the method of (Uribe et al. 2021) can be applied because they are rare-event problems (i.e., ϕ is of the form $\phi = \mathbb{I}_{\{\varphi \geq 0\}}$). In the next section we present other applications where the evaluation of the FIS is not feasible since either the function is not differentiable (test case of Section 5.4) or the example is not a rare event simulation problem (test cases of Section 5.3 and Section 5.5).

5.3 Test case 3: banana shape distribution

The third test case we consider is the integration of the banana shape distribution h, which is a classical test case in importance sampling (Cornuet et al. 2012), (Elvira et al. 2019). The banana shape distribution is the following pdf

$$h(\mathbf{x}) = g_{\mathbf{0},C}(x_1, x_2 + b(x_1^2 - \sigma^2), x_3, \dots, x_n). \tag{12}$$

The term $g_{\mathbf{0},C}$ represents the pdf of a Gaussian distribution of mean $\mathbf{0}$ and diagaonal covariance matrix $C = \text{diag}(\sigma^2, 1, \dots, 1)$. The value of b and σ^2 are respectively set to $b = 800$ and $\sigma^2 = 0.0025$. We choose ϕ such that the optimal IS density g^* is equal to h, i.e., we choose $\phi = h/f$ so that the integral $\mathscr{E}$ that we are trying to estimate is equal to $\mathscr{E} = \int \phi f = 1$. This choice is made in order to have an optimal covariance matrix * whose two largest eigenvalues (in ℓ-order) correspond to the smallest and largest eigenvalues, as can be seen in Figure 4b. More formally, the optimal value of the Gaussian parameters are given by $\mathbf{m}^* = \mathbf{0}$ and * is diagonal with

$$^* = \begin{pmatrix} 0.0025 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 9 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

The evolution of the KL partial divergence is given in Figure 4a. As the optimal mean $\mathbf{m}^*$ is equal to $\mathbf{0}$, we cannot project on $\mathbf{m}^*$ and so the matrix$\hat{}_{\text{mean}}$ is not defined. However, the numerical estimation $\widehat{\mathbf{m}}^*$ will not be equal to 0 and so the approach proposed in (El Masri, Morio, and Simatos 2021) with $\hat{}^{+\text{d}}_{\text{mean}}$ is still applicable numerically.

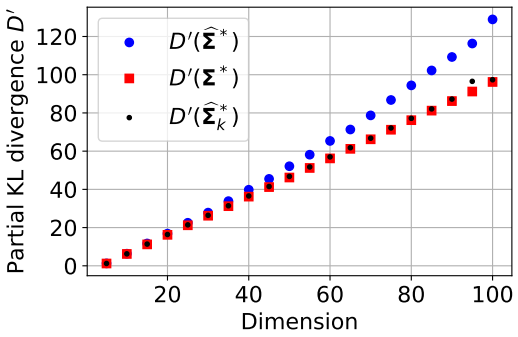
The simulation results for the different covariance matrices and the vMFN density are given in Table 4. The matrices$\hat{}_{\text{opt}}$ and$\hat{}^{+\text{d}}_{\text{opt}}$ perform very well for the estimation of E with an accuracy of the same order as the optimal covariance matrix *. The effect of estimating the $k = 2$ main projection directions does not affect much the estimation performance as$\hat{}^{+\text{d}}_{\text{opt}}$ is still efficient compared to$\hat{}_{\text{opt}}$. The estimation results with$\hat{}^{+\text{d}}_{\text{mean}}$ are not really accurate and this choice is in fact roughly equivalent to choosing a random projection direction. The vMFN parametric model is not adapted to this test case as the vMFN estimate is not close to 1.

```
################################################################################
# Figure 4. Evolution of the partial KL divergence and spectrum of the
# eigenvalues for the test case 3
################################################################################

b=800
s2=0.0025
```

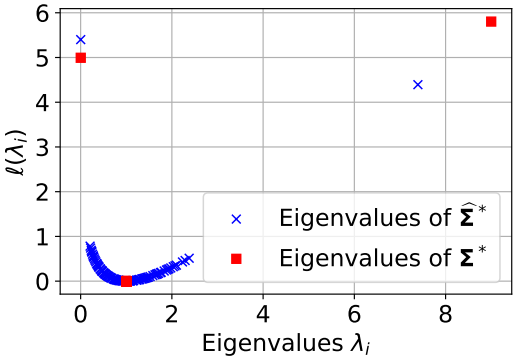
```
def bananapdf(X):
    XX=np.copy(X)
    n=np.shape(XX)[1]
    I=np.eye(n)
    I[0,0]=s2
    XX[:,1]=XX[:,1]+b*(XX[:,0]**2-s2)
    f=sp.stats.multivariate_normal.pdf(XX,mean=np.zeros(n),cov=I)
    return(f)


DKL=np.zeros(20)
DKLp=np.zeros(20)
DKLm=np.zeros(20)
DKLstar=np.zeros(20)

n=100
M=300

for d in range(5,n+1,5):

    I=np.eye(d)
    I[0,0]=s2

    #Mstar
    Mstar = np.zeros(d)
    #Sigmastar
    Sigstar=np.copy(I)
    Sigstar[1,1]=9       #1+2*b^2*s2^2


    ## g*-sample
    X=sp.stats.multivariate_normal.rvs(mean=np.zeros(d),cov=I,size=M)
    X[:,1]=X[:,1]-b*(X[:,0]**2-s2)

    ## estimated mean and covariance
    mm=np.mean(X,axis=0)

    Xc=(X-mm).T
    sigma=Xc @ Xc.T/np.shape(Xc)[1]

    ## projection with the eigenvalues of sigma
    Eig=np.linalg.eigh(sigma)
    logeig=np.sort(np.log(Eig[0])-Eig[0])
    delta=np.zeros(len(logeig)-1)
    for j in range(len(logeig)-1):
        delta[j]=abs(logeig[j]-logeig[j+1])

    k=np.argmax(delta)+1         # biggest gap between the l(lambda_i)
```

```
    indi=[]
    for l in range(k):
        indi.append(np.where(np.log(Eig[0])-Eig[0]==logeig[l])[0][0])

    P1=np.array(Eig[1][:,indi[0]],ndmin=2).T                    # projection matrix
    for l in range(1,k):
        P1=np.concatenate((P1,np.array(Eig[1][:,indi[l]],ndmin=2).T),axis=1)

    diagsi=np.diag(Eig[0][indi])
    sig_opt_d=P1.dot((diagsi-np.eye(k))).dot(P1.T)+np.eye(d)

    DKL[int((d-5)/5)]=np.log(np.linalg.det(sigma))+np.sum(np.diag(\
                                    Sigstar.dot(np.linalg.inv(sigma))))
    DKLp[int((d-5)/5)]=np.log(np.linalg.det(sig_opt_d))+np.sum(np.diag(\
                                    Sigstar.dot(np.linalg.inv(sig_opt_d))))
    DKLstar[int((d-5)/5)]=np.log(np.linalg.det(Sigstar))+d

#### plot of partial KL divergence
plt.plot(range(5,n+1,5),DKL,'bo',label=r"$D'(\widehat{\mathbf{\Sigma}}^*)$")
plt.plot(range(5,n+1,5),DKLstar,'rs',label=r"$D'(\mathbf{\Sigma}^*)$")
plt.plot(range(5,n+1,5),DKLp,'k.',label=r"$D'(\widehat{\mathbf{\Sigma}}^*_k)$")



plt.grid()
plt.xlabel('Dimension',fontsize=16)
plt.ylabel(r"Partial KL divergence $D'$",fontsize=16)
plt.legend(fontsize=16)
for tickLabel in plt.gca().get_xticklabels() + plt.gca().get_yticklabels():
    tickLabel.set_fontsize(16)
plt.show()

#### plot of the eigenvalues
Eig1=np.linalg.eigh(sigma)
logeig1=np.log(Eig1[0])-Eig1[0]+1
Table_eigv=np.zeros((n,2))
Table_eigv[:,0]=Eig1[0]
Table_eigv[:,1]=-logeig1

Eigst=np.linalg.eigh(Sigstar)
logeigst=np.log(Eigst[0])-Eigst[0]+1
Table_eigv_st=np.zeros((n,2))
Table_eigv_st[:,0]=Eigst[0]
Table_eigv_st[:,1]=-logeigst

plt.grid()
plt.xlabel(r"Eigenvalues $\lambda_i$",fontsize=16)
plt.ylabel(r"$\ell(\lambda_i)$",fontsize=16)
for tickLabel in plt.gca().get_xticklabels() + plt.gca().get_yticklabels():
```

```
    tickLabel.set_fontsize(16)

plt.plot(Table_eigv[:,0],Table_eigv[:,1],'bx',\
         label=r"Eigenvalues of $\widehat{\mathbf{\Sigma}}^*$")
plt.plot(Table_eigv_st[:,0],Table_eigv_st[:,1],'rs',\
         label=r"Eigenvalues of $\mathbf{\Sigma}^*$")
plt.legend(fontsize=16)
plt.show()
```



(a) Evolution of the partial KL divergence as the dimension increases, with the optimal covariance matrix * (red saquares), the sample covariance$\hat{}^*$ (blue circles), and the projected covariance$\hat{}_k^*$ (black dots).



(b) Computation of $\ell(\lambda_i)$ for the eigenvalues of * (red squares) and$\hat{}^*$ (blue crosses) in dimension $n = 100$ for the banana shape example of Equation 12.

Figure 4: Partial KL divergence and spectrum for the banana shape example.

```
################################################################################
# Table 4. Numerical comparison on test case 3
################################################################################

n=100          # dimension
phi=bananapdf
E=1

mypi=bananapdf

N=2000
M=500
B=500    # number of runs

Eopt=np.zeros(B)
EIS=np.zeros(B)
Eprj=np.zeros(B)
Eprm=np.zeros(B)
Eprjst=np.zeros(B)
Eprmst=np.zeros(B)
Evmfn=np.zeros(B)
```

```
SI=[]
SIP=[]
SIPst=[]
SIM=[]
SIMst=[]

I=np.eye(d)
I[0,0]=s2

#Mstar
Mstar = np.zeros(d)
#Sigmastar
Sigstar=np.copy(I)
Sigstar[1,1]=9       #1+2*b^2*s2^2

Eigst=np.linalg.eigh(Sigstar)
logeigst=np.sort(np.log(Eigst[0])-Eigst[0])
deltast=np.zeros(len(logeigst)-1)

for i in range(len(logeigst)-1):
    deltast[i]=abs(logeigst[i]-logeigst[i+1])

## choice of the number of dimension
k_st=np.argmax(deltast)+1

indist=[]
for i in range(k_st):
    indist.append(np.where(np.log(Eigst[0])-Eigst[0]==logeigst[i])[0][0])

P1st=np.array(Eigst[1][:,indist[0]],ndmin=2).T
for i in range(1,k_st):
    # matrix of influential directions
    P1st=np.concatenate((P1st,np.array(Eigst[1][:,indist[i]],ndmin=2).T)\
                        ,axis=1)

#np.random.seed(0)
for i in range(B):
############################## Estimation of the matrices

   ## g*-sample of size M
    X=sp.stats.multivariate_normal.rvs(mean=np.zeros(d),cov=I,size=M)
    X[:,1]=X[:,1]-b*(X[:,0]**2-s2)

   ## estimated mean and covariance
    mm=np.mean(X,axis=0)
    Xc=(X-mm).T
    sigma=Xc @ Xc.T/np.shape(Xc)[1]
    SI.append(sigma)
```

```
    R=np.sqrt(np.sum(X**2,axis=1))
    Xu=(X.T/R).T

   ## von Mises Fisher parameters
    normu=np.sqrt(np.mean(Xu,axis=0).dot(np.mean(Xu,axis=0).T))
    mu=np.mean(Xu,axis=0)/normu
    mu=np.array(mu,ndmin=2)
    chi=min(normu,0.95)
    kappa=(chi*n-chi**3)/(1-chi**2)

   ## Nakagami parameters
    omega=np.mean(R**2)
    tau4=np.mean(R**4)
    pp=omega**2/(tau4-omega**2)

   ###
    Eig=np.linalg.eigh(sigma)
    logeig=np.sort(np.log(Eig[0])-Eig[0])
    delta=np.zeros(len(logeig)-1)
    for j in range(len(logeig)-1):
        delta[j]=abs(logeig[j]-logeig[j+1])

    k=np.argmax(delta)+1

    indi=[]
    for l in range(k):
        indi.append(np.where(np.log(Eig[0])-Eig[0]==logeig[l])[0][0])

    P1=np.array(Eig[1][:,indi[0]],ndmin=2).T
    for l in range(1,k):
        P1=np.concatenate((P1,np.array(Eig[1][:,indi[l]],ndmin=2).T),axis=1)

    diagsi=np.diag(Eig[0][indi])
    sig_opt_d=P1.dot((diagsi-np.eye(k))).dot(P1.T)+np.eye(n)
    SIP.append(sig_opt_d)

   ###
    diagsist=P1st.T.dot(sigma).dot(P1st)
    sig_opt=P1st.dot(diagsist-np.eye(k_st)).dot(P1st.T)+np.eye(n)
    SIPst.append(sig_opt)

   ###
    Norm_mm=np.linalg.norm(mm)
    normalised_mm=np.array(mm,ndmin=2).T/Norm_mm
    vhat=normalised_mm.T.dot(sigma).dot(normalised_mm)
    sig_mean_d=(vhat-1)*normalised_mm.dot(normalised_mm.T)+np.eye(n)
    SIM.append(sig_mean_d)

############################################## Estimation of the integral
```

```
    ###
     Xop=sp.stats.multivariate_normal.rvs(mean=mm, cov=Sigstar,size=N)
     wop=mypi(Xop)/sp.stats.multivariate_normal.pdf(Xop,mean=mm, cov=Sigstar)
     Eopt[i]=np.mean(wop)

    ###
     Xis=sp.stats.multivariate_normal.rvs(mean=mm, cov=sigma,size=N)
     wis=mypi(Xis)/sp.stats.multivariate_normal.pdf(Xis,mean=mm, cov=sigma)
     EIS[i]=np.mean(wis)

    ###
     Xpr=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt_d,size=N)
     wpr=mypi(Xpr)/sp.stats.multivariate_normal.pdf(Xpr,mean=mm, \
                                                    cov=sig_opt_d)
     Eprj[i]=np.mean(wpr)

    ###
     Xpm=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_mean_d,size=N)
     wpm=mypi(Xpm)/sp.stats.multivariate_normal.pdf(Xpm,mean=mm, \
                                                    cov=sig_mean_d)
     Eprm[i]=np.mean(wpm)

    ###
     Xprst=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt,size=N)
     wprst=mypi(Xprst)/sp.stats.multivariate_normal.pdf(Xprst,mean=mm, \
                                                        cov=sig_opt)
     Eprjst[i]=np.mean(wprst)

     Xvmfn = vMFNM_sample(mu, kappa, omega, pp, 1, N)
     Rvn=np.sqrt(np.sum(Xvmfn**2,axis=1))
     Xvnu=Xvmfn.T/Rvn

     h_log=vMF_logpdf(Xvnu,mu.T,kappa)+nakagami_logpdf(Rvn,pp,omega)
     A = np.log(n) + np.log(np.pi ** (n / 2)) - sp.special.gammaln(n / 2 + 1)
     f_u = -A
     f_chi = (np.log(2) * (1 - n / 2) + np.log(Rvn) * (n - 1) - 0.5 * \
              Rvn ** 2 - sp.special.gammaln(n / 2))
     f_log = f_u + f_chi
     W_log = f_log - h_log

     wvmfn=(phi(Xvmfn)>0)*np.exp(W_log)
     Evmfn[i]=np.mean(wvmfn)

### KL divergences
dkli=np.zeros(B)
dklp=np.zeros(B)
dklm=np.zeros(B)
dklpst=np.zeros(B)
```

```
for i in range(B):
    dkli[i]=np.log(np.linalg.det(SI[i]))+sum(np.diag(\
                                        Sigstar.dot(np.linalg.inv(SI[i]))))
    dklp[i]=np.log(np.linalg.det(SIP[i]))+sum(np.diag(\
                                        Sigstar.dot(np.linalg.inv(SIP[i]))))
    dklm[i]=np.log(np.linalg.det(SIM[i]))+sum(np.diag(\
                                        Sigstar.dot(np.linalg.inv(SIM[i]))))
    dklpst[i]=np.log(np.linalg.det(SIPst[i]))+sum(np.diag(\
                                        Sigstar.dot(np.linalg.inv(SIPst[i]))))

Tabresult=np.zeros((3,7)) # table of results

Tabresult[0,0]=np.log(np.linalg.det(Sigstar))+n
Tabresult[0,1]=np.mean(dkli)
Tabresult[0,2]=np.mean(dklpst)
Tabresult[0,3]=None
Tabresult[0,4]=np.mean(dklp)
Tabresult[0,5]=np.mean(dklm)
Tabresult[0,6]=None

Tabresult[1,0]=np.mean(Eopt-E)/E*100
Tabresult[1,1]=np.mean(EIS-E)/E*100
Tabresult[1,2]=np.mean(Eprjst-E)/E*100
Tabresult[1,3]=None
Tabresult[1,4]=np.mean(Eprj-E)/E*100
Tabresult[1,5]=np.mean(Eprm-E)/E*100
Tabresult[1,6]=np.mean(Evmfn-E)/E*100

Tabresult[2,0]=np.sqrt(np.mean((Eopt-E)**2))/E*100
Tabresult[2,1]=np.sqrt(np.mean((EIS-E)**2))/E*100
Tabresult[2,2]=np.sqrt(np.mean((Eprjst-E)**2))/E*100
Tabresult[2,3]=None
Tabresult[2,4]=np.sqrt(np.mean((Eprj-E)**2))/E*100
Tabresult[2,5]=np.sqrt(np.mean((Eprm-E)**2))/E*100
Tabresult[2,6]=np.sqrt(np.mean((Evmfn-E)**2))/E*100

Tabresult=np.round(Tabresult,1)

table=[["D'",Tabresult[0,0],Tabresult[0,1],Tabresult[0,2],"NA",
        Tabresult[0,4],Tabresult[0,5],"/"],
      ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],
       Tabresult[1,2],"NA",Tabresult[1,4],Tabresult[1,5],Tabresult[1,6]],
    ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],
     Tabresult[2,2],"NA",Tabresult[2,4],Tabresult[2,5],Tabresult[2,6]]]
Markdown(tabulate(
  table,
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}
           "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
           "$\widehat{\mathbf{\Sigma}}^{+d}_{mean}$", "vMFN"],
```

```
    tablefmt="pipe"))
```

```
<>:210: SyntaxWarning: invalid escape sequence '\%'
<>:212: SyntaxWarning: invalid escape sequence '\%'
<>:216: SyntaxWarning: invalid escape sequence '\m'
<>:216: SyntaxWarning: invalid escape sequence '\w'
<>:216: SyntaxWarning: invalid escape sequence '\w'
<>:217: SyntaxWarning: invalid escape sequence '\w'
<>:217: SyntaxWarning: invalid escape sequence '\w'
<>:218: SyntaxWarning: invalid escape sequence '\w'
<>:210: SyntaxWarning: invalid escape sequence '\%'
<>:212: SyntaxWarning: invalid escape sequence '\%'
<>:216: SyntaxWarning: invalid escape sequence '\m'
<>:216: SyntaxWarning: invalid escape sequence '\w'
<>:216: SyntaxWarning: invalid escape sequence '\w'
<>:217: SyntaxWarning: invalid escape sequence '\w'
<>:217: SyntaxWarning: invalid escape sequence '\w'
<>:218: SyntaxWarning: invalid escape sequence '\w'
/tmp/ipykernel_7763/605444355.py:210: SyntaxWarning: invalid escape sequence '\%'
  ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],
/tmp/ipykernel_7763/605444355.py:212: SyntaxWarning: invalid escape sequence '\%'
  ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],
/tmp/ipykernel_7763/605444355.py:216: SyntaxWarning: invalid escape sequence '\m'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/605444355.py:216: SyntaxWarning: invalid escape sequence '\w'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/605444355.py:216: SyntaxWarning: invalid escape sequence '\w'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/605444355.py:217: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
/tmp/ipykernel_7763/605444355.py:217: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
/tmp/ipykernel_7763/605444355.py:218: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}^{+d}_{mean}$", "vMFN"],
```

Table 4: Numerical comparison of the estimation of $\mathscr{E} = 1$ considering the Gaussian density with the six covariance matrices defined in Section 4.2 and the vFMN model, $\phi = h/f$. NA stands for non applicable, as explained in the text. The computational cost is $N = 2000$.

	$*$	$\hat{}*$	$\hat{}_{opt}$	$\hat{}_{mean}$	$\hat{}^{+d}_{opt}$	$\hat{}^{+d}_{mean}$	vMFN
D'	96.2	110.8	96.2	NA	96.8	106.8	/
Relative error (%)	-0.6	3.5	-1.1	NA	-2.2	-10.4	- 83.0
Coefficient of variation (%)	8.6	593.2	6.7	NA	8	35.7	83.0

5.4 Application 1: large portfolio losses

The next example is a rare event application in finance, taken from (Bassamboo, Juneja, and Zeevi 2008), (Chan and Kroese 2012). The unknown integral is $\mathscr{E} = \int_{\mathbb{R}^{n+2}} \phi(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} = \mathbb{P}_f(\varphi(\mathbf{X}) \geq 0)$, with $\phi = \mathbb{I}_{\{\varphi \geq 0\}}$ and f is the standard $n+2$-dimensional Gaussian distribution. The function φ is the portfolio loss function defined as:

$$\varphi(\mathbf{x}) = \sum_{j=3}^{n+2} \mathbb{I}_{\{\Psi(x_1,x_2,x_j) \geq 0.5\sqrt{n}\}} - bn, \tag{13}$$

with

$$\Psi(x_1, x_2, x_j) = \left(q x_1 + 3(1-q^2)^{1/2} x_j\right) \left[F_\Gamma^{-1}\left(F_{\mathscr{N}}(x_2)\right)\right]^{-1/2},$$

where F_Γ and $F_{\mathscr{N}}$ are the cumulative distribution functions of Gamma$(6,6)$ and $\mathscr{N}(0,1)$ random variables respectively. The constant b is choosen such that the probability is of the order of 10^{-3} in all dimension, then we have $b = 0.45$ when $n \leq 30$, $b = 0.3$ when $30 < n \leq 70$, and $b = 0.25$ when $n > 70$.

The reference value of this probability $\mathscr{E}$ is reported in Table 5 for dimension $n = 100$. The optimal parameters $\mathbf{m}^*$ and * cannot be computed analytically, but they are accurately estimated by Monte Carlo with a large sample. It turns out that $\mathbf{m}^*$ and the first eigenvector $\mathbf{d}_1^*$ of * are numerically indistinguishable and that Algorithm 2 selects $k = 1$ projection direction, so that numerically, the choices$\hat{}_{\text{opt}}$ and$\hat{}_{\text{mean}}$ are indistinguishable and gives the same estimation results. Actually, the fact that these two estimators behave similarly does not seem to come from the fact that $\mathbf{m}^*$ and $\mathbf{d}^*$ are close: this relation can be broken for instance by a simple translation argument (see remark after Table 6), but even then they behave similarly. The KL partial divergence and the spectrum with the associated ℓ-order are presented respectively in Figure 5a and in Figure 5b.

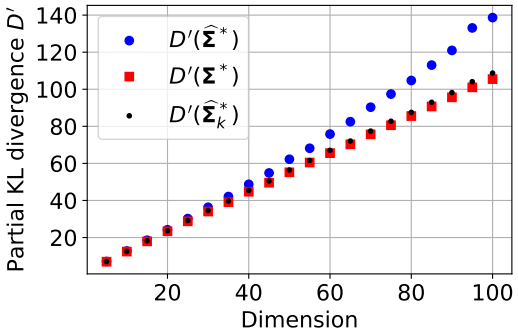
```
##############################################################################
# Figure 5. Evolution of the partial KL divergence and spectrum of the
# eigenvalues for the large portfolio loss application
##############################################################################

def Portfolio(X):
    N=np.shape(X)[0]
    nn=np.shape(X)[1]
    n=nn-2
    lamb=np.array(sp.stats.gamma.ppf(sp.stats.norm.cdf(X[:,0]),6,scale=1/6)\
                  ,ndmin=2).T
    eta=3*X[:,2:]
    ZZ=np.array(X[:,1],ndmin=2).T

    XX=(1/4*ZZ+np.sqrt(1-1/4**2)*eta)/np.sqrt(lamb)
    IndX=(XX>0.5*np.sqrt(n))*1
    PF=np.sum(IndX,axis=1)
    return(PF-0.25*n-0.1)

def Portfolio_md(X):
    N=np.shape(X)[0]
    nn=np.shape(X)[1]
    n=nn-2
    lamb=np.array( sp.stats.gamma.ppf(sp.stats.norm.cdf(X[:,0]),6,scale=1/6)\
```

```
                    ,ndmin=2).T
    eta=3*X[:,2:]
    ZZ=np.array(X[:,1],ndmin=2).T

    XX=(1/4*ZZ+np.sqrt(1-1/4**2)*eta)/np.sqrt(lamb)
    IndX=(XX>0.5*np.sqrt(n))*1
    PF=np.sum(IndX,axis=1)
    return(PF-0.3*n-0.1)

def Portfolio_ld(X):
    N=np.shape(X)[0]
    nn=np.shape(X)[1]
    n=nn-2
    lamb=np.array(sp.stats.gamma.ppf(sp.stats.norm.cdf(X[:,0]),6,scale=1/6)\
                    ,ndmin=2).T
    eta=3*X[:,2:]
    ZZ=np.array(X[:,1],ndmin=2).T

    XX=(1/4*ZZ+np.sqrt(1-1/4**2)*eta)/np.sqrt(lamb)
    IndX=(XX>0.5*np.sqrt(n))*1
    PF=np.sum(IndX,axis=1)
    return(PF-0.45*n-0.1)


DKL=np.zeros(20)
DKLp=np.zeros(20)
DKLm=np.zeros(20)
DKLstar=np.zeros(20)

n=100
bigsample=20*10**5
M=300

for d in range(5,n+1,5):

    if d<=30:
        phi=Portfolio_ld
    if d>70:
        phi=Portfolio
    else:
        phi=Portfolio_md

    VA=sp.stats.multivariate_normal(mean=np.zeros(d+2),cov=np.eye(d+2))
    XO1=VA.rvs(size=bigsample)
    ind1=(phi(XO1)>0)
    X1=XO1[ind1,:]
    X1=X1[:M*10,:]
    #Mstar
    Mstar=np.mean(X1.T,axis=1)
```

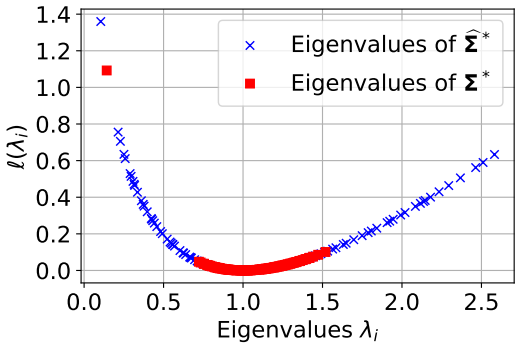
```
    #Sigmastar
    X1c=(X1-Mstar).T
    Sigstar=X1c.dot(X1c.T)/np.shape(X1c)[1]


    ## g*-sample
    VA0=sp.stats.multivariate_normal(mean=np.zeros(d+2),cov=np.eye(d+2))
    X0=VA0.rvs(size=M*1000)

    ind=(phi(X0)>0)
    X=X0[ind,:]
    X=X[:M,:]             # g*-sample of size M

    ## estimated mean and covariance
    mm=np.mean(X,axis=0)

    Xc=(X-mm).T
    sigma =Xc @ Xc.T/np.shape(Xc)[1]

    ## projection with the eigenvalues of sigma
    Eig=np.linalg.eigh(sigma)
    logeig=np.sort(np.log(Eig[0])-Eig[0])
    delta=np.zeros(len(logeig)-1)
    for j in range(len(logeig)-1):
        delta[j]=abs(logeig[j]-logeig[j+1])

    k=np.argmax(delta)+1         # biggest gap between the l(lambda_i)

    indi=[]
    for l in range(k):
        indi.append(np.where(np.log(Eig[0])-Eig[0]==logeig[l])[0][0])

    P1=np.array(Eig[1][:,indi[0]],ndmin=2).T          # projection matrix
    for l in range(1,k):
        P1=np.concatenate((P1,np.array(Eig[1][:,indi[l]],ndmin=2).T),axis=1)

    diagsi=np.diag(Eig[0][indi])
    sig_opt_d=P1.dot((diagsi-np.eye(k))).dot(P1.T)+np.eye(d+2)

    DKL[int((d-5)/5)]=np.log(np.linalg.det(sigma))+np.sum(np.diag(\
                                    Sigstar.dot(np.linalg.inv(sigma))))
    DKLp[int((d-5)/5)]=np.log(np.linalg.det(sig_opt_d))+np.sum(np.diag(\
                                    Sigstar.dot(np.linalg.inv(sig_opt_d))))
    DKLstar[int((d-5)/5)]=np.log(np.linalg.det(Sigstar))+d+2

#### plot of partial KL divergence
plt.plot(range(5,n+1,5),DKL,'bo',label=r"$D'(\widehat{\mathbf{\Sigma}}^*)$")
plt.plot(range(5,n+1,5),DKLstar,'rs',label=r"$D'(\mathbf{\Sigma}^*)$")
plt.plot(range(5,n+1,5),DKLp,'k.',label=r"$D'(\widehat{\mathbf{\Sigma}}^*_k)$")
```

```
plt.grid()
plt.xlabel('Dimension',fontsize=16)
plt.ylabel(r"Partial KL divergence $D'$",fontsize=16)
plt.legend(fontsize=16)
for tickLabel in plt.gca().get_xticklabels() + plt.gca().get_yticklabels():
    tickLabel.set_fontsize(16)
plt.show()

#### plot of the eigenvalues
Eig1=np.linalg.eigh(sigma)
logeig1=np.log(Eig1[0])-Eig1[0]+1
Table_eigv=np.zeros((n+2,2))
Table_eigv[:,0]=Eig1[0]
Table_eigv[:,1]=-logeig1

Eigst=np.linalg.eigh(Sigstar)
logeigst=np.log(Eigst[0])-Eigst[0]+1
Table_eigv_st=np.zeros((n+2,2))
Table_eigv_st[:,0]=Eigst[0]
Table_eigv_st[:,1]=-logeigst

plt.grid()
plt.xlabel(r"Eigenvalues $\lambda_i$",fontsize=16)
plt.ylabel(r"$\ell(\lambda_i)$",fontsize=16)
for tickLabel in plt.gca().get_xticklabels() + plt.gca().get_yticklabels():
    tickLabel.set_fontsize(16)

plt.plot(Table_eigv[:,0],Table_eigv[:,1],'bx',\
         label=r"Eigenvalues of $\widehat{\mathbf{\Sigma}}^*$")
plt.plot(Table_eigv_st[:,0],Table_eigv_st[:,1],'rs',\
         label=r"Eigenvalues of $\mathbf{\Sigma}^*$")
plt.legend(fontsize=16)
plt.show()
```

```
################################################################################
# Table 5. Numerical comparison on the large portfolio loss application
################################################################################

n=100         # dimension
phi=Portfolio
E=1.82*10**(-3)

def mypi(X):
    nn=np.shape(X)[1]
    n=nn-2
    f0=sp.stats.multivariate_normal.pdf(X,mean=np.zeros(nn),cov=np.eye(nn))
    return((phi(X)>0)*f0)
```



(a) Evolution of the partial KL divergence as the dimension increases, with the optimal covariance matrix * (red squares), the sample covariance$\hat{}^*$ (blue circles), and the projected covariance$\hat{}^*_k$ (black dots).



(b) Computation of $\ell(\lambda_i)$ for the eigenvalues of * (red squares) and$\hat{}^*$ (blue crosses) in dimension $n = 100$ for the large portfolio losses of Equation 13.

Figure 5: Partial KL divergence and spectrum for the function $\phi = \mathbb{I}_{\varphi \geq 0}$ with φ the function given by Equation 13.

```
N=2000
M=500
B=500   # number of runs

Eopt=np.zeros(B)
EIS=np.zeros(B)
Eprj=np.zeros(B)
Eprm=np.zeros(B)
Eprjst=np.zeros(B)
Eprmst=np.zeros(B)
Evmfn=np.zeros(B)

SI=[]
SIP=[]
SIPst=[]
SIM=[]
SIMst=[]


### Mstar and Sigmastar have been estimated offline with
### a 10^6 Monte Carlo sample from g^*
#Mstar
Mstar=pickle.load( open( "Mstar_portfolio.p", "rb" ) )
#Sigmastar
Sigstar=pickle.load( open( "Sigstar_portfolio.p", "rb" ) )

Eigst=np.linalg.eigh(Sigstar)
logeigst=np.sort(np.log(Eigst[0])-Eigst[0])
deltast=np.zeros(len(logeigst)-1)
```

```
for i in range(len(logeigst)-1):
    deltast[i]=abs(logeigst[i]-logeigst[i+1])

## choice of the number of dimension
k_st=np.argmax(deltast)+1

indist=[]
for i in range(k_st):
    indist.append(np.where(np.log(Eigst[0])-Eigst[0]==logeigst[i])[0][0])

P1st=np.array(Eigst[1][:,indist[0]],ndmin=2).T
for i in range(1,k_st):
    # matrix of influential directions
    P1st=np.concatenate((P1st,np.array(Eigst[1][:,indist[i]],ndmin=2).T),\
                        axis=1)

#np.random.seed(0)
for i in range(B):
############################## Estimation of the matrices

   ## g*-sample of size M
    VA=sp.stats.multivariate_normal(np.zeros(n+2),np.eye(n+2))
    X0=VA.rvs(size=M*1000)
    ind=(phi(X0)>0)
    X=X0[ind,:]
    X=X[:M,:]

    R=np.sqrt(np.sum(X**2,axis=1))
    Xu=(X.T/R).T

   ## estimated gaussian mean and covariance
    mm=np.mean(X,axis=0)
    Xc=(X-mm).T
    sigma =Xc @ Xc.T/np.shape(Xc)[1]
    SI.append(sigma)

   ## von Mises Fisher parameters
    normu=np.sqrt(np.mean(Xu,axis=0).dot(np.mean(Xu,axis=0).T))
    mu=np.mean(Xu,axis=0)/normu
    mu=np.array(mu,ndmin=2)
    chi=min(normu,0.95)
    kappa=(chi*n-chi**3)/(1-chi**2)

   ## Nakagami parameters
    omega=np.mean(R**2)
    tau4=np.mean(R**4)
    pp=omega**2/(tau4-omega**2)

   ###
```

```
    Eig=np.linalg.eigh(sigma)
    logeig=np.sort(np.log(Eig[0])-Eig[0])
    delta=np.zeros(len(logeig)-1)
    for j in range(len(logeig)-1):
        delta[j]=abs(logeig[j]-logeig[j+1])

    k=np.argmax(delta)+1

    indi=[]
    for l in range(k):
        indi.append(np.where(np.log(Eig[0])-Eig[0]==logeig[l])[0][0])

    P1=np.array(Eig[1][:,indi[0]],ndmin=2).T
    for l in range(1,k):
        P1=np.concatenate((P1,np.array(Eig[1][:,indi[l]],ndmin=2).T),axis=1)

    diagsi=np.diag(Eig[0][indi])
    sig_opt_d=P1.dot((diagsi-np.eye(k))).dot(P1.T)+np.eye(n+2)
    SIP.append(sig_opt_d)

   ###
    diagsist=P1st.T.dot(sigma).dot(P1st)
    sig_opt=P1st.dot(diagsist-np.eye(k_st)).dot(P1st.T)+np.eye(n+2)
    SIPst.append(sig_opt)

   ###
    Norm_mm=np.linalg.norm(mm)
    normalised_mm=np.array(mm,ndmin=2).T/Norm_mm
    vhat=normalised_mm.T.dot(sigma).dot(normalised_mm)
    sig_mean_d=(vhat-1)*normalised_mm.dot(normalised_mm.T)+np.eye(n+2)
    SIM.append(sig_mean_d)

   ###
    Norm_Mstar=np.linalg.norm(Mstar)
    normalised_Mstar=np.array(Mstar,ndmin=2).T/Norm_Mstar
    vhatst=normalised_Mstar.T.dot(sigma).dot(normalised_Mstar)

    sig_mean=(vhatst-1)*normalised_Mstar.dot(normalised_Mstar.T)+np.eye(n+2)
    SIMst.append(sig_mean)

############################################## Estimation of the integral
   ###
    Xop=sp.stats.multivariate_normal.rvs(mean=mm, cov=Sigstar,size=N)
    wop=mypi(Xop)/sp.stats.multivariate_normal.pdf(Xop,mean=mm, cov=Sigstar)
    Eopt[i]=np.mean(wop)

   ###
    Xis=sp.stats.multivariate_normal.rvs(mean=mm, cov=sigma,size=N)
    wis=mypi(Xis)/sp.stats.multivariate_normal.pdf(Xis,mean=mm, cov=sigma)
```

```
    EIS[i]=np.mean(wis)

    ###
    Xpr=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt_d,size=N)
    wpr=mypi(Xpr)/sp.stats.multivariate_normal.pdf(Xpr,mean=mm, \
                                                   cov=sig_opt_d)
    Eprj[i]=np.mean(wpr)

    ###
    Xpm=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_mean_d,size=N)
    wpm=mypi(Xpm)/sp.stats.multivariate_normal.pdf(Xpm,mean=mm, \
                                                   cov=sig_mean_d)
    Eprm[i]=np.mean(wpm)

    ###
    Xprst=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt,size=N)
    wprst=mypi(Xprst)/sp.stats.multivariate_normal.pdf(Xprst,mean=mm, \
                                                       cov=sig_opt)
    Eprjst[i]=np.mean(wprst)

    ###
    Xpmst=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_mean,size=N)
    wpmst=mypi(Xpmst)/sp.stats.multivariate_normal.pdf(Xpmst,mean=mm, \
                                                       cov=sig_mean)
    Eprmst[i]=np.mean(wpmst)

    ###
    Xvmfn = vMFNM_sample(mu, kappa, omega, pp, 1, N)
    Rvn=np.sqrt(np.sum(Xvmfn**2,axis=1))
    Xvnu=Xvmfn.T/Rvn

    h_log=vMF_logpdf(Xvnu,mu.T,kappa)+nakagami_logpdf(Rvn,pp,omega)
    A = np.log(n+2) + np.log(np.pi ** ((n+2) / 2)) - sp.special.gammaln((n+2) / 2 + 1)
    f_u = -A
    f_chi = (np.log(2) * (1 - (n+2) / 2) + np.log(Rvn) * ((n+2) - 1)\
             - 0.5 * Rvn ** 2 - sp.special.gammaln((n+2) / 2))
    f_log = f_u + f_chi
    W_log = f_log - h_log

    wvmfn=(phi(Xvmfn)>0)*np.exp(W_log)
    Evmfn[i]=np.mean(wvmfn)

### KL divergences
dkli=np.zeros(B)
dklp=np.zeros(B)
dklm=np.zeros(B)
dklpst=np.zeros(B)
dklmst=np.zeros(B)
```

```
for i in range(B):
    dkli[i]=np.log(np.linalg.det(SI[i]))+sum(np.diag(\
                        Sigstar.dot(np.linalg.inv(SI[i]))))
    dklp[i]=np.log(np.linalg.det(SIP[i]))+sum(np.diag(\
                        Sigstar.dot(np.linalg.inv(SIP[i]))))
    dklm[i]=np.log(np.linalg.det(SIM[i]))+sum(np.diag(\
                        Sigstar.dot(np.linalg.inv(SIM[i]))))
    dklpst[i]=np.log(np.linalg.det(SIPst[i]))+sum(np.diag(\
                        Sigstar.dot(np.linalg.inv(SIPst[i]))))
    dklmst[i]=np.log(np.linalg.det(SIMst[i]))+sum(np.diag(\
                        Sigstar.dot(np.linalg.inv(SIMst[i]))))

Tabresult=np.zeros((3,7)) # table of results

Tabresult[0,0]=np.log(np.linalg.det(Sigstar))+n+2
Tabresult[0,1]=np.mean(dkli)
Tabresult[0,2]=np.mean(dklpst)
Tabresult[0,3]=np.mean(dklmst)
Tabresult[0,4]=np.mean(dklp)
Tabresult[0,5]=np.mean(dklm)
Tabresult[0,6]=None

Tabresult[1,0]=np.mean(Eopt-E)/E*100
Tabresult[1,1]=np.mean(EIS-E)/E*100
Tabresult[1,2]=np.mean(Eprjst-E)/E*100
Tabresult[1,3]=np.mean(Eprmst-E)/E*100
Tabresult[1,4]=np.mean(Eprj-E)/E*100
Tabresult[1,5]=np.mean(Eprm-E)/E*100
Tabresult[1,6]=np.mean(Evmfn-E)/E*100

Tabresult[2,0]=np.sqrt(np.mean((Eopt-E)**2))/E*100
Tabresult[2,1]=np.sqrt(np.mean((EIS-E)**2))/E*100
Tabresult[2,2]=np.sqrt(np.mean((Eprjst-E)**2))/E*100
Tabresult[2,3]=np.sqrt(np.mean((Eprmst-E)**2))/E*100
Tabresult[2,4]=np.sqrt(np.mean((Eprj-E)**2))/E*100
Tabresult[2,5]=np.sqrt(np.mean((Eprm-E)**2))/E*100
Tabresult[2,6]=np.sqrt(np.mean((Evmfn-E)**2))/E*100

Tabresult=np.round(Tabresult,1)

table=[["D'",Tabresult[0,0],Tabresult[0,1],Tabresult[0,2],Tabresult[0,3],
        Tabresult[0,4],Tabresult[0,5],"/"],
      ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],
       Tabresult[1,2],Tabresult[1,3],Tabresult[1,4],Tabresult[1,5],Tabresult[1,6]],
    ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],
     Tabresult[2,2],Tabresult[2,3],Tabresult[2,4],Tabresult[2,5],Tabresult[2,6]]]
Markdown(tabulate(
  table,
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}
```

```
            "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
            "$\widehat{\mathbf{\Sigma}}^{+d}_{mean}$", "vMFN"],
      tablefmt="pipe"))
```

```
<>:235: SyntaxWarning: invalid escape sequence '\%'
<>:237: SyntaxWarning: invalid escape sequence '\%'
<>:241: SyntaxWarning: invalid escape sequence '\m'
<>:241: SyntaxWarning: invalid escape sequence '\w'
<>:241: SyntaxWarning: invalid escape sequence '\w'
<>:242: SyntaxWarning: invalid escape sequence '\w'
<>:242: SyntaxWarning: invalid escape sequence '\w'
<>:243: SyntaxWarning: invalid escape sequence '\w'
<>:235: SyntaxWarning: invalid escape sequence '\%'
<>:237: SyntaxWarning: invalid escape sequence '\%'
<>:241: SyntaxWarning: invalid escape sequence '\m'
<>:241: SyntaxWarning: invalid escape sequence '\w'
<>:241: SyntaxWarning: invalid escape sequence '\w'
<>:242: SyntaxWarning: invalid escape sequence '\w'
<>:242: SyntaxWarning: invalid escape sequence '\w'
<>:243: SyntaxWarning: invalid escape sequence '\w'
/tmp/ipykernel_7763/3889756799.py:235: SyntaxWarning: invalid escape sequence '\%'
  ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],
/tmp/ipykernel_7763/3889756799.py:237: SyntaxWarning: invalid escape sequence '\%'
  ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],
/tmp/ipykernel_7763/3889756799.py:241: SyntaxWarning: invalid escape sequence '\m'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/3889756799.py:241: SyntaxWarning: invalid escape sequence '\w'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/3889756799.py:241: SyntaxWarning: invalid escape sequence '\w'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/3889756799.py:242: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
/tmp/ipykernel_7763/3889756799.py:242: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
/tmp/ipykernel_7763/3889756799.py:243: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}^{+d}_{mean}$", "vMFN"],
```

Table 5: Numerical comparison of the estimation of $\mathscr{E} \approx 1.82 \cdot 10^{-3}$ considering the Gaussian density with the six covariance matrices defined in Section 4.2 and the vFMN model, $\phi = \mathbb{I}_{\varphi \geq 0}$ with φ given by Equation 13. The computational cost is $N = 2000$.

	$*$	$\widehat{*}$	$\widehat{}_{opt}$	$\widehat{}_{mean}$	$\widehat{}^{+d}_{opt}$	$\widehat{}^{+d}_{mean}$	vMFN
D'	107.3	122.5	107.6	107.6	108	107.7	/
Relative error (%)	0.6	0.4	0.6	-0.2	0.4	0.9	0.4
Coefficient of variation (%)	6.5	370.1	9.7	7.5	12.2	14.9	6.5

The results of Table 5 show similar trends as for the first test case of Section 5.1. First, projecting seems indeed a relevant idea, as using $\hat{}_{\text{opt}}$ or $\hat{}_{\text{mean}}$ greatly improves the situation compared to $\hat{}^*$. This is particularly salient as $\hat{}^*$ yields an important bias and a coefficient of variation of 370%, whereas projecting on $\mathbf{d}_1^*$ or $\mathbf{m}^*$ yields a coefficient of variation of 12% and of 7.5% respectively. This improvement is still true even when the projection directions are estimated: compared to $\hat{}_{\text{opt}}$ and $\hat{}_{\text{mean}}$, $\hat{}_{\text{opt}}^{+\text{d}}$ and $\hat{}_{\text{mean}}^{+\text{d}}$ give coefficients of variation between 12.2 and 14.9%. Finally, $\hat{}_{\text{opt}}^{+\text{d}}$ seems to behave better than $\hat{}_{\text{mean}}^{+\text{d}}$.

5.5 Application 2: discretized Asian payoff

Our last numerical experiment is a mathematical finance example coming from (Kawai 2018), representing a discrete approximation of a standard Asian payoff under the Black–Scholes model. The goal is to estimate the integral $\mathscr{E} = \int_{\mathbb{R}^n} \phi(\mathbf{x}) f(\mathbf{x}) \mathrm{d}\mathbf{x}$ with f the standard n-dimensional Gaussian distribution and the following function ϕ:

$$\phi : \mathbf{x} = (x_1, \dots, x_n) \mapsto e^{-rT} \left[\frac{S_0}{n} \sum_{i=1}^{n} \exp\left(i \left(r - \frac{\sigma^2}{2} \right) \frac{T}{n} + \sigma \sqrt{\frac{T}{n}} \sum_{k=1}^{i} x_k \right) - K \right]_+ \tag{14}$$

where $[y]_+ = \max(y, 0)$, for a real number y. The constants are taken from (Kawai 2018): $S_0 = 50$, $r = 0.05, T = 0.5, \sigma = 0.1, K = 55$, where they test the function for dimension $n = 16$. In our contribution, we test this example in dimension 100. Concerning $\mathbf{m}^*$ and the $\mathbf{d}_i^*$'s, the situation is the same as in the previous example: they are not available analytically but can be estimated numerically by Monte Carlo with a large simulation budget. And again, it turns out that $\mathbf{m}^*$ and the first eigenvector $\mathbf{d}_1^*$ of * are numerically indistinguishable and that Algorithm 2 selects $k = 1$ projection direction, so that $\hat{}_{\text{opt}}$ and $\hat{}_{\text{mean}}$ yield results that are numerically indistinguishable. The KL partial divergence and the spectrum with the associated ℓ-order are respectively presented in Figure 6a and Figure 6b.

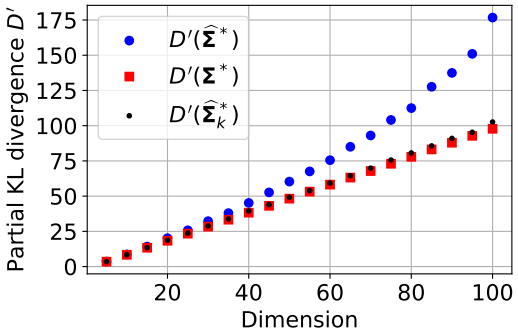
The results of this example are given in Table 6. The insight gained in the previous examples is confirmed. Projecting on $\mathbf{m}^*$ or $\mathbf{d}_1^*$ in dimension $n = 100$ enables to reach convergence and reduces (compared to $\hat{}^*$) the coefficient of variation from 559% to nearly 2%. Moreover, this improvement goes through even when projection directions are estimated, with again $\hat{}_{\text{mean}}^{+\text{d}}$ behaving better than $\hat{}_{\text{opt}}^{+\text{d}}$.

```
################################################################################
# Figure 6. Evolution of the partial KL divergence and spectrum of the
# eigenvalues for the asian payoff application
################################################################################

def payoff(X):
    d=np.shape(X)[1]
    S0=50
    r=0.05
    T=0.5
    sig2=0.01
    K=55
    uk=(r-sig2/2)*T/d+np.sqrt(T*sig2/d)*X
    cumuk=np.cumsum(uk,axis=1)
    en=S0*np.exp(cumuk)
    FK=np.exp(-r*T)*(1/d*np.sum(en,axis=1)-K)
    return(FK*(FK>0))
```

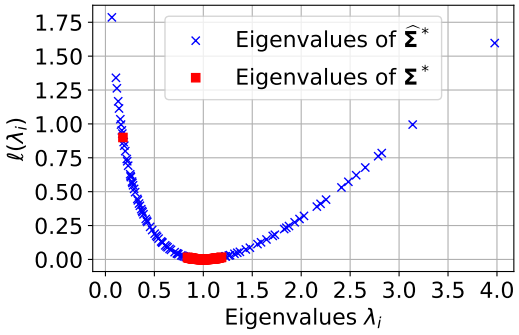
```
DKL=np.zeros(20)
DKLp=np.zeros(20)
DKLm=np.zeros(20)
DKLstar=np.zeros(20)

n=100
M=300
bigsample=10*10**5
phi=payoff

for d in range(5,n+1,5):

    VA=sp.stats.multivariate_normal(mean=np.zeros(d),cov=np.eye(d))
    X1=VA.rvs(size=bigsample)
    W1=phi(X1)
    W=W1[(W1>0)]
    X=X1[(W1>0),:]
#     W=W[:10*M]
#     X=X[:10*M,:]

    ## Mstar
    Mstar = np.divide((W.T @ X), sum(W))
    ## Sigmastar
    Xc = np.multiply((X - Mstar).T, np.sqrt(W))
    Sigstar = np.divide((Xc @ Xc.T), sum(W))

    ##
    VA0=sp.stats.multivariate_normal(np.zeros(d),np.eye(d))
    X0=VA0.rvs(size=M*100)
    W0=phi(X0)
    Wf=W0[(W0>0)]
    Xf=X0[(W0>0),:]
    Wf=Wf[:M]
    Xf=Xf[:M,:]

    ## estimated mean and covariance
    mm=np.divide((Wf.T @ Xf), sum(Wf))
    Xcf=np.multiply((Xf - mm).T, np.sqrt(Wf))
    sigma =np.divide((Xcf @ Xcf.T), sum(Wf))


    ## projection with the eigenvalues of sigma
    Eig=np.linalg.eigh(sigma)
    logeig=np.sort(np.log(Eig[0])-Eig[0])
    delta=np.zeros(len(logeig)-1)
    for j in range(len(logeig)-1):
        delta[j]=abs(logeig[j]-logeig[j+1])
```

```
    k=np.argmax(delta)+1          # biggest gap between the l(lambda_i)

    indi=[]
    for l in range(k):
        indi.append(np.where(np.log(Eig[0])-Eig[0]==logeig[l])[0][0])

    P1=np.array(Eig[1][:,indi[0]],ndmin=2).T          # projection matrix
    for l in range(1,k):
        P1=np.concatenate((P1,np.array(Eig[1][:,indi[l]],ndmin=2).T),axis=1)

    diagsi=np.diag(Eig[0][indi])
    sig_opt_d=P1.dot((diagsi-np.eye(k))).dot(P1.T)+np.eye(d)

    DKL[int((d-5)/5)]=np.log(np.linalg.det(sigma))+np.sum(\
                            np.diag(Sigstar.dot(np.linalg.inv(sigma))))
    DKLp[int((d-5)/5)]=np.log(np.linalg.det(sig_opt_d))+np.sum(\
                            np.diag(Sigstar.dot(np.linalg.inv(sig_opt_d))))
    DKLstar[int((d-5)/5)]=np.log(np.linalg.det(Sigstar))+d

#### plot of partial KL divergence
plt.plot(range(5,n+1,5),DKL,'bo',label=r"$D'(\widehat{\mathbf{\Sigma}}^*)$")
plt.plot(range(5,n+1,5),DKLstar,'rs',label=r"$D'(\mathbf{\Sigma}^*)$")
plt.plot(range(5,n+1,5),DKLp,'k.',label=r"$D'(\widehat{\mathbf{\Sigma}}^*_k)$")

plt.grid()
plt.xlabel('Dimension',fontsize=16)
plt.ylabel(r"Partial KL divergence $D'$",fontsize=16)
plt.legend(fontsize=16)
for tickLabel in plt.gca().get_xticklabels() + plt.gca().get_yticklabels():
    tickLabel.set_fontsize(16)
plt.show()

#### plot of the eigenvalues
Eig1=np.linalg.eigh(sigma)
logeig1=np.log(Eig1[0])-Eig1[0]+1
Table_eigv=np.zeros((n,2))
Table_eigv[:,0]=Eig1[0]
Table_eigv[:,1]=-logeig1

Eigst=np.linalg.eigh(Sigstar)
logeigst=np.log(Eigst[0])-Eigst[0]+1
Table_eigv_st=np.zeros((n,2))
Table_eigv_st[:,0]=Eigst[0]
Table_eigv_st[:,1]=-logeigst

plt.grid()
plt.xlabel(r"Eigenvalues $\lambda_i$",fontsize=16)
plt.ylabel(r"$\ell(\lambda_i)$",fontsize=16)
```

```
for tickLabel in plt.gca().get_xticklabels() + plt.gca().get_yticklabels():
    tickLabel.set_fontsize(16)

plt.plot(Table_eigv[:,0],Table_eigv[:,1],'bx',\
         label=r"Eigenvalues of $\widehat{\mathbf{\Sigma}}^*$")
plt.plot(Table_eigv_st[:,0],Table_eigv_st[:,1],'rs',\
         label=r"Eigenvalues of $\mathbf{\Sigma}^*$")
plt.legend(fontsize=16)
plt.show()
```



(a) Evolution of the partial KL divergence as the dimension increases, with the optimal covariance matrix * (red squares), the sample covariance$\hat{}^*$ (blue circles), and the projected covariance$\hat{}^*_k$ (black dots).



(b) Computation of $\ell(\lambda_i)$ for the eigenvalues of * (red squares) and$\hat{}^*$ (blue crosses) in dimension $n = 100$ for the Asian payoff example of Equation 14

Figure 6: Partial KL divergence and spectrum for the function ϕ given in Equation 14.

```
##############################################################################
# Table 6. Numerical comparison on the Asian payoff application
##############################################################################

n=100          # dimension

bigsample=2*10**6
phi=payoff
E=0.0187

def mypi(X):
    n=np.shape(X)[1]
    return(sp.stats.multivariate_normal.pdf(X,mean=np.zeros(n),\
                                            cov=np.eye(n))*phi(X))

N=2000
M=500
B=500     # number of runs

### Mstar and Sigmastar have been estimated offline with
### a 10^6 Monte Carlo sample from g^*
```

```
#Mstar
Mstar=pickle.load( open( "Mstar_asian.p", "rb" ) )
#Sigmastar
Sigstar=pickle.load( open( "Sigstar_asian.p", "rb" ) )

Eigst=np.linalg.eigh(Sigstar)
logeigst=np.sort(np.log(Eigst[0])-Eigst[0])
deltast=np.zeros(len(logeigst)-1)

for l in range(len(logeigst)-1):
    deltast[l]=abs(logeigst[l]-logeigst[l+1])

## choice of the number of dimension
k_st=np.argmax(deltast)+1

indist=[]
for j in range(k_st):
    indist.append(np.where(np.log(Eigst[0])-Eigst[0]==logeigst[j])[0][0])

P1st=np.array(Eigst[1][:,indist[0]],ndmin=2).T
for jj in range(1,k_st):
    # matrix of influential directions
    P1st=np.concatenate((P1st,np.array(Eigst[1][:,\
        indist[jj]],ndmin=2).T),axis=1)

Eopt=np.zeros(B)
EIS=np.zeros(B)
Eprj=np.zeros(B)
Eprm=np.zeros(B)
Eprjst=np.zeros(B)
Eprmst=np.zeros(B)
Evmfn=np.zeros(B)

SI=[]
SIP=[]
SIPst=[]
SIM=[]
SIMst=[]

#np.random.seed(0)
for i in range(B):
############################## Estimation of the matrices

   ##
    VA0=sp.stats.multivariate_normal(mean=np.zeros(n),cov=np.eye(n))
    X0=VA.rvs(size=100*M)
    W0=phi(X0)
    Wf=W0[(W0>0)]
    Xf=X0[(W0>0),:]
```

```
 Wf=Wf[:M]
 Xf=Xf[:M,:]

## estimated mean and covariance
 mm=np.divide((Wf.T @ Xf), sum(Wf))
 Xcf=np.multiply((Xf - mm).T, np.sqrt(Wf))
 sigma=np.divide((Xcf @ Xcf.T), sum(Wf))
 SI.append(sigma)

 R=np.sqrt(np.sum(Xf**2,axis=1))
 Xu=(Xf.T/R).T

## von Mises Fisher parameters
 normu=np.sqrt(np.mean(Xu,axis=0).dot(np.mean(Xu,axis=0).T))
 mu=np.mean(Xu,axis=0)/normu
 mu=np.array(mu,ndmin=2)
 chi=min(normu,0.95)
 kappa=(chi*n-chi**3)/(1-chi**2)

## Nakagami parameters
 omega=np.mean(R**2)
 tau4=np.mean(R**4)
 pp=omega**2/(tau4-omega**2)

###
 Eig=np.linalg.eigh(sigma)
 logeig=np.sort(np.log(Eig[0])-Eig[0])
 delta=np.zeros(len(logeig)-1)
 for j in range(len(logeig)-1):
     delta[j]=abs(logeig[j]-logeig[j+1])

 k=np.argmax(delta)+1

 indi=[]
 for l in range(k):
     indi.append(np.where(np.log(Eig[0])-Eig[0]==logeig[l])[0][0])

 P1=np.array(Eig[1][:,indi[0]],ndmin=2).T
 for l in range(1,k):
     P1=np.concatenate((P1,np.array(Eig[1][:,indi[l]],ndmin=2).T),axis=1)

 diagsi=np.diag(Eig[0][indi])
 sig_opt_d=P1.dot((diagsi-np.eye(k))).dot(P1.T)+np.eye(n)
 SIP.append(sig_opt_d)

###
 diagsist=P1st.T.dot(sigma).dot(P1st)
 sig_opt=P1st.dot(diagsist-np.eye(k_st)).dot(P1st.T)+np.eye(n)
 SIPst.append(sig_opt)
```

```
    ###
     Norm_mm=np.linalg.norm(mm)
     normalised_mm=np.array(mm,ndmin=2).T/Norm_mm
     vhat=normalised_mm.T.dot(sigma).dot(normalised_mm)
     sig_mean_d=(vhat-1)*normalised_mm.dot(normalised_mm.T)+np.eye(n)
     SIM.append(sig_mean_d)

    ###
     Norm_Mstar=np.linalg.norm(Mstar)
     normalised_Mstar=np.array(Mstar,ndmin=2).T/Norm_Mstar
     vhatst=normalised_Mstar.T.dot(sigma).dot(normalised_Mstar)

     sig_mean=(vhatst-1)*normalised_Mstar.dot(normalised_Mstar.T)+np.eye(n)
     SIMst.append(sig_mean)

############################################## Estimation of the integral
    ###
     Xop=sp.stats.multivariate_normal.rvs(mean=mm, cov=Sigstar,size=N)
     wop=mypi(Xop)/sp.stats.multivariate_normal.pdf(Xop,mean=mm, cov=Sigstar)
     Eopt[i]=np.mean(wop)

    ###
     Xis=sp.stats.multivariate_normal.rvs(mean=mm, cov=sigma,size=N)
     wis=mypi(Xis)/sp.stats.multivariate_normal.pdf(Xis,mean=mm, cov=sigma)
     EIS[i]=np.mean(wis)

    ###
     Xpr=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt_d,size=N)
     wpr=mypi(Xpr)/sp.stats.multivariate_normal.pdf(Xpr,mean=mm, \
                                                    cov=sig_opt_d)
     Eprj[i]=np.mean(wpr)

    ###
     Xpm=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_mean_d,size=N)
     wpm=mypi(Xpm)/sp.stats.multivariate_normal.pdf(Xpm,mean=mm, \
                                                    cov=sig_mean_d)
     Eprm[i]=np.mean(wpm)

    ###
     Xprst=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt,size=N)
     wprst=mypi(Xprst)/sp.stats.multivariate_normal.pdf(Xprst,mean=mm, \
                                                        cov=sig_opt)
     Eprjst[i]=np.mean(wprst)

    ###
     Xpmst=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_mean,size=N)
     wpmst=mypi(Xpmst)/sp.stats.multivariate_normal.pdf(Xpmst,mean=mm, \
                                                        cov=sig_mean)
```

```
    Eprmst[i]=np.mean(wpmst)

   ###
    Xvmfn = vMFNM_sample(mu, kappa, omega, pp, 1, N)
    Rvn=np.sqrt(np.sum(Xvmfn**2,axis=1))
    Xvnu=Xvmfn.T/Rvn

    h_log=vMF_logpdf(Xvnu,mu.T,kappa)+nakagami_logpdf(Rvn,pp,omega)
    A = np.log(n) + np.log(np.pi ** (n / 2)) - sp.special.gammaln(n / 2 + 1)
    f_u = -A
    f_chi = (np.log(2) * (1 - n / 2) + np.log(Rvn) * (n - 1) - 0.5\
             * Rvn ** 2 - sp.special.gammaln(n / 2))
    f_log = f_u + f_chi
    W_log = f_log - h_log

    wvmfn=(phi(Xvmfn)>0)*np.exp(W_log)
    Evmfn[i]=np.mean(wvmfn)

### KL divergences
dkli=np.zeros(B)
dklp=np.zeros(B)
dklm=np.zeros(B)
dklpst=np.zeros(B)
dklmst=np.zeros(B)

for i in range(B):
    dkli[i]=np.log(np.linalg.det(SI[i]))+sum(np.diag(\
        Sigstar.dot(np.linalg.inv(SI[i]))))
    dklp[i]=np.log(np.linalg.det(SIP[i]))+sum(np.diag(\
        Sigstar.dot(np.linalg.inv(SIP[i]))))
    dklm[i]=np.log(np.linalg.det(SIM[i]))+sum(np.diag(\
        Sigstar.dot(np.linalg.inv(SIM[i]))))
    dklpst[i]=np.log(np.linalg.det(SIPst[i]))+sum(np.diag(\
        Sigstar.dot(np.linalg.inv(SIPst[i]))))
    dklmst[i]=np.log(np.linalg.det(SIMst[i]))+sum(np.diag(\
        Sigstar.dot(np.linalg.inv(SIMst[i]))))

Tabresult=np.zeros((3,7)) # table of results

Tabresult[0,0]=np.log(np.linalg.det(Sigstar))+n
Tabresult[0,1]=np.mean(dkli)
Tabresult[0,2]=np.mean(dklpst)
Tabresult[0,3]=np.mean(dklmst)
Tabresult[0,4]=np.mean(dklp)
Tabresult[0,5]=np.mean(dklm)
Tabresult[0,6]=None

Tabresult[1,0]=np.mean(Eopt-E)/E*100
Tabresult[1,1]=np.mean(EIS-E)/E*100
```

```
Tabresult[1,2]=np.mean(Eprjst-E)/E*100
Tabresult[1,3]=np.mean(Eprmst-E)/E*100
Tabresult[1,4]=np.mean(Eprj-E)/E*100
Tabresult[1,5]=np.mean(Eprm-E)/E*100
Tabresult[1,6]=np.mean(Evmfn-E)/E*100

Tabresult[2,0]=np.sqrt(np.mean((Eopt-E)**2))/E*100
Tabresult[2,1]=np.sqrt(np.mean((EIS-E)**2))/E*100
Tabresult[2,2]=np.sqrt(np.mean((Eprjst-E)**2))/E*100
Tabresult[2,3]=np.sqrt(np.mean((Eprmst-E)**2))/E*100
Tabresult[2,4]=np.sqrt(np.mean((Eprj-E)**2))/E*100
Tabresult[2,5]=np.sqrt(np.mean((Eprm-E)**2))/E*100
Tabresult[2,6]=np.sqrt(np.mean((Evmfn-E)**2))/E*100

Tabresult=np.round(Tabresult,1)


table=[["D'",Tabresult[0,0],Tabresult[0,1],Tabresult[0,2],Tabresult[0,3],
        Tabresult[0,4],Tabresult[0,5],"/"],
      ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],
       Tabresult[1,2],Tabresult[1,3],Tabresult[1,4],Tabresult[1,5],Tabresult[1,6]],
    ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],
     Tabresult[2,2],Tabresult[2,3],Tabresult[2,4],Tabresult[2,5],Tabresult[2,6]]]
Markdown(tabulate(
  table,
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}
           "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
           "$\widehat{\mathbf{\Sigma}}^{+d}_{mean}$", "vMFN"],
    tablefmt="pipe"))
```

```
<>:237: SyntaxWarning: invalid escape sequence '\%'
<>:239: SyntaxWarning: invalid escape sequence '\%'
<>:243: SyntaxWarning: invalid escape sequence '\m'
<>:243: SyntaxWarning: invalid escape sequence '\w'
<>:243: SyntaxWarning: invalid escape sequence '\w'
<>:244: SyntaxWarning: invalid escape sequence '\w'
<>:244: SyntaxWarning: invalid escape sequence '\w'
<>:245: SyntaxWarning: invalid escape sequence '\w'
<>:237: SyntaxWarning: invalid escape sequence '\%'
<>:239: SyntaxWarning: invalid escape sequence '\%'
<>:243: SyntaxWarning: invalid escape sequence '\m'
<>:243: SyntaxWarning: invalid escape sequence '\w'
<>:243: SyntaxWarning: invalid escape sequence '\w'
<>:244: SyntaxWarning: invalid escape sequence '\w'
<>:244: SyntaxWarning: invalid escape sequence '\w'
<>:245: SyntaxWarning: invalid escape sequence '\w'
/tmp/ipykernel_7763/842017826.py:237: SyntaxWarning: invalid escape sequence '\%'
  ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],
/tmp/ipykernel_7763/842017826.py:239: SyntaxWarning: invalid escape sequence '\%'
  ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],
```

```
/tmp/ipykernel_7763/842017826.py:243: SyntaxWarning: invalid escape sequence '\m'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/842017826.py:243: SyntaxWarning: invalid escape sequence '\w'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/842017826.py:243: SyntaxWarning: invalid escape sequence '\w'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/842017826.py:244: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
/tmp/ipykernel_7763/842017826.py:244: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
/tmp/ipykernel_7763/842017826.py:245: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}^{+d}_{mean}$", "vMFN"],
```

Table 6: Numerical comparison of the estimation of $\mathscr{E} \approx 18.7 \times 10^{-3}$ considering the Gaussian density with the six covariance matrices defined in Section 4.2 and the vFMN model, when ϕ is given by Equation 14. The computational cost is $N = 2000$.

	$\mathbf{\Sigma}^*$	$\widehat{\mathbf{\Sigma}}^*$	$\widehat{\mathbf{\Sigma}}_{opt}$	$\widehat{\mathbf{\Sigma}}_{mean}$	$\widehat{\mathbf{\Sigma}}^{+d}_{opt}$	$\widehat{\mathbf{\Sigma}}^{+d}_{mean}$	vMFN
D'	98.3	127.9	98.3	98.3	99.5	98.5	/
Relative error (%)	0.4	-37.2	0.4	0.3	-0.3	0.3	18.3
Coefficient of variation (%)	2.2	559.2	2.6	3.1	14.2	2.7	18.9

Remark. As already mentioned, the two directions $\mathbf{m}^*$ and $\mathbf{d}_1^*$ are numerically indistinguishable in the two examples of Section 5.4 and Section 5.5. However, we do not believe this relation to be highly relevant. For instance, this symmetry can be broken by changing ϕ into $\phi' = \phi(\cdot - \mu)$ and f into $f' = f(\cdot - \mu)$ for some $\mu \in \mathbb{R}^n$. Since $g^* \propto \phi f$, this amounts to translating g^* which thus changes $\mathbf{m}^*$ into $\mathbf{m}^{*\prime} = \mathbf{m}^* + \mu$, but which does not change the covariance matrix (and therefore its leading eigenvector $\mathbf{d}_1^*$) which is translation-invariant. Note that this translation does not affect the integral $\mathscr{E} = \int \phi' f' = \int \phi f$, and so this modification leads to a new estimator $\widehat{\mathscr{E}}_\mu$ of the same quantity $\mathscr{E}$. However, it can be shown that $\widehat{\mathscr{E}}_\mu$ and $\widehat{\mathscr{E}}$ (the estimators considered throughout the paper) have the same law so that this translation, although it does break the relation $\mathbf{m}^* \approx \mathbf{d}_1^*$, does not change the law of the estimators. This suggests that, if the estimators based on $\widehat{\mathbf{\Sigma}}_{\text{opt}}$ and $\widehat{\mathbf{\Sigma}}_{\text{mean}}$ do behave similarly on these examples, this is not due to the fact that $\mathbf{m}^*$ and $\mathbf{d}_1^*$ are close but rather to Theorem 3.1 and Theorem 3.2. However, the fact that $\mathbf{m}^*$ and $\mathbf{d}_1^*$ are close bears some insight into the importance of the quality of the estimation of the projection direction as we now elaborate in the conclusion.

6 Conclusion

The goal of this paper is to assess the efficiency of projection methods in order to overcome the curse of dimensionality for importance sampling. Based on a new theoretical result (Theorem 3.1), we propose to project on the subspace spanned by the eigenvectors $\mathbf{d}_i^*$'s corresponding to the largest eigenvalues of the optimal covariance matrix $\mathbf{\Sigma}^*$, where eigenvalues are ranked based on their image by some explicit function ℓ. Our numerical results show that if the $\mathbf{d}_i^*$'s were perfectly known, then projecting on them (column $\widehat{\mathbf{\Sigma}}_{\text{opt}}$ in the result tables of Section 5) would greatly improve the final

estimation compared to using the empirical estimation of the covariance matrix (column$\hat{}^*$) and actually lead to results which are comparable to those obtained with the optimal covariance matrix (column *). Moreover, we show that this improvement goes through when one estimates the $\mathbf{d}_i^*$'s (column$\hat{}^{+\mathrm{d}}_{\mathrm{opt}}$) by computing the eigenpairs of$\hat{}^*$: indeed, using$\hat{}^{+\mathrm{d}}_{\mathrm{opt}}$ as covariance matrix instead of$\hat{}^*$ gives results which remain accurate up to the dimension $n = 100$ considered in the present paper.

Moreover, we compare the directions $\mathbf{d}_i^*$'s, which are justified by Theorem 3.1, with the algorithm proposed in (El Masri, Morio, and Simatos 2021) which amounts to projecting on $\mathbf{m}^*$ (column$\hat{}_{\mathrm{mean}}$ when $\mathbf{m}^*$ is assumed to be known, or column$\hat{}^{+\mathrm{d}}_{\mathrm{mean}}$ when one uses the estimation $\widehat{\mathbf{m}}^*$ of $\mathbf{m}^*$). On three out of the five numerical examples considered, it turns out that $\mathbf{m}^*$ and $\mathbf{d}_1^*$ can be proved to be equal, or are numerically indistinguishable, and Algorithm 2 selects one projection direction. In these cases, the choices$\hat{}_{\mathrm{opt}}$ and$\hat{}_{\mathrm{mean}}$ lead to similar numerical results. The second and third test cases of Section 5.2 break the relation between $\mathbf{m}^*$ and $\mathbf{d}_1^*$, and in this case,$\hat{}^{+\mathrm{d}}_{\mathrm{opt}}$ clearly outperforms $\hat{}^{+\mathrm{d}}_{\mathrm{mean}}$. It would be interesting to delve deeper into the relation between $\mathbf{m}^*$ and the eigenvectors of *, and try and understand when estimating the $\mathbf{d}_i^*$'s instead of the simpler $\mathbf{m}^*$ is indeed worthwhile.

These theoretical and numerical results show that the $\mathbf{d}_i^*$'s of Theorem 3.1 are good directions in which to estimate variance terms. With the insight gained, we see several ways to extend our results. Two in particular stand out:

- study different ways of estimating the eigenpairs $(\lambda_i^*, \mathbf{d}_i^*)$;
- incorporate this method in adaptive importance sampling schemes, in particular the cross-entropy method (Rubinstein and Kroese 2017b).

For the first point, remember that we made the choice to estimate the eigenpairs of * by computing the eigenpairs of$\hat{}^*$. Moreover, in the numerical examples of Section 5.1, Section 5.4 and Section 5.5 where $\mathbf{m}^*$ and $\mathbf{d}_1^*$ are equal or indistinguishable, we saw that$\hat{}^{+\mathrm{d}}_{\mathrm{mean}}$ performed better than$\hat{}^{+\mathrm{d}}_{\mathrm{opt}}$ and we conjecture that this is because $\widehat{\mathbf{m}}$ is a better estimator than $\hat{\mathbf{d}}_1^*$ (recall that $\mathbf{m}^* = \mathbf{d}_1^*$ for the example of Section 5.1, while in Section 5.4 and Section 5.5 they are numerically indistinguishable and so, for all practical purposes, $\widehat{\mathbf{m}}^*$ and $\hat{\mathbf{d}}_1^*$ can be considered estimators of the same direction). This suggests that improving the estimation of the $\mathbf{d}_i^*$'s can indeed improve the final estimation of $\mathscr{E}$. Possible ways to do so consist in adapting existing results on the estimation of covariance matrices (for instance (Ledoit and Wolf 2004)) or even directly results on the estimation of eigenvalues of covariance matrices such as (Benaych-Georges and Nadakuditi 2011), (Mestre 2008a), (Mestre 2008b), (Nadakuditi and Edelman 2008), which we plan to do in future work. Moreover, it would be interesting to relax the assumption that one can sample from g^* in order to estimate$\hat{}^*$. For the second point, we plan to investigate how the idea of the present paper can improve the efficiency of adaptive importance sampling schemes in high dimensions. In this case, there is an additional difficulty, namely the introduction of likelihood ratios can lead to the problem of weight degeneracy which is another reason why performance of such schemes degrades in high dimensions (Bengtsson, Bickel, and Li 2008}).

We note finally that it would be interesting to consider multimodal failure functions ϕ. Indeed, with unimodal functions, the light tail of the Gaussian random variable implies that the conditional variance decreases which explains why, in all our numerical examples with an indicator function, the highest eigenvalues ranked in ℓ-order are simply the smallest eigenvalues. However, for multimodal failure functions, we may expect the conditional variance to increase and that the highest eigenvalues ranked in ℓ-order are actually the largest ones. For multimodal problems, one may want to consider different parametric families of auxiliary densities, and so it would be interesting to see whether Theorem 3.1 can be extended to more general cases.

Acknowledgement

The first author was enrolled in a PhD program co-funded by ISAE-SUPAERO and ONERA—The French Aerospace Lab. Their financial support is gratefully acknowledged. This work is part of the activities of ONERA - ISAE - ENAC joint research group.

Appendix A: Proof of Theorem 3.1 and Theorem 3.2

We begin with a preliminary lemma.

Lemma 6.1. *Let f be the density of the standard Gaussian vector in dimension n, $\phi : \mathbb{R}^n \to \mathbb{R}_+$ and $g_* = f\phi/\mathscr{E}$ with $\mathscr{E} = \int f\phi$. Then for any $\mathbf{m}$ and any $$ of the form $ = I_n + \sum_i (\alpha_i - 1)\mathbf{d}_i\mathbf{d}_i^\top$ with $\alpha_i > 0$ and the $\mathbf{d}_i$'s orthonormal, we have*

$$
\begin{aligned}
D(g^*, g_{\mathbf{m},}) = & \frac{1}{2}\sum_i \left(\log\alpha_i - \left(1 - \frac{1}{\alpha_i}\right)\mathbf{d}_i^{\top *}\mathbf{d}_i\right) + \frac{1}{2}(\mathbf{m}-\mathbf{m}^*)^{\top -1}(\mathbf{m}-\mathbf{m}^*) \\
& - \frac{1}{2}\|\mathbf{m}^*\|^2 - \log\mathscr{E} + \mathbb{E}_{g^*}(\log\phi(\mathbf{X})).
\end{aligned}
\tag{15}
$$

Proof of Lemma 6.1

For any $\mathbf{m} \in \mathbb{R}^n$ and $ \in \mathcal{S}_n^+$, we have by definition

$$
D(g^*, g_{\mathbf{m},}) = \mathbb{E}_{g^*}\left(\log\left(\frac{g^*(\mathbf{X})}{g_{\mathbf{m},}(\mathbf{X})}\right)\right) = \mathbb{E}_{g^*}\left(\log\left(\frac{\frac{\phi(\mathbf{X})e^{-\frac{1}{2}\|\mathbf{X}\|^2}}{\mathscr{E}(2\pi)^{n/2}}}{\frac{e^{-\frac{1}{2}(\mathbf{X}-\mathbf{m})^{\top -1}(\mathbf{X}-\mathbf{m})}}{(2\pi)^{n/2}||^{1/2}}}\right)\right)
$$

and so

$$
\begin{aligned}
D(g^*, g_{\mathbf{m},}) = & -\frac{1}{2}\mathbb{E}_{g^*}(\|\mathbf{X}\|^2) + \frac{1}{2}\mathbb{E}_{g^*}\left((\mathbf{X}-\mathbf{m})^{\top -1}(\mathbf{X}-\mathbf{m})\right) \\
& + \frac{1}{2}\log|| - \log\mathscr{E} + \mathbb{E}_{g^*}(\log\phi(\mathbf{X})).
\end{aligned}
$$

Because $\mathbb{E}_{g^*}(\mathbf{X}) = \mathbf{m}^*$, we have $\mathbb{E}_{g^*}(\|\mathbf{X}\|^2) = \mathbb{E}_{g^*}(\|\mathbf{X}-\mathbf{m}^*\|^2) + \|\mathbf{m}^*\|^2$ and

$$
\mathbb{E}_{g^*}\left((\mathbf{X}-\mathbf{m})^{\top -1}(\mathbf{X}-\mathbf{m})\right) = \mathbb{E}_{g^*}\left((\mathbf{X}-\mathbf{m}^*)^{\top -1}(\mathbf{X}-\mathbf{m}^*)\right) + (\mathbf{m}-\mathbf{m}^*)^{\top -1}(\mathbf{m}-\mathbf{m}^*).
$$

In the following derivations, we use the linearity of the trace and of the expectation, which make these two operators commute, as well as the identity $a^\top b = \mathrm{tr}(ab^\top)$ for any two vectors a and b. With this caveat, we obtain

$$
\mathbb{E}_{g^*}\left[\|\mathbf{X}-\mathbf{m}^*\|^2\right] = \mathbb{E}_{g^*}\left[\mathrm{tr}((\mathbf{X}-\mathbf{m}^*)(\mathbf{X}-\mathbf{m}^*)^\top)\right] = \mathrm{tr}(^*)
$$

and we obtain with similar arguments $\mathbb{E}_{g^*}((\mathbf{X}-\mathbf{m}^*)^{\top -1}(\mathbf{X}-\mathbf{m}^*)) = \mathrm{tr}(^{-1*})$. Consider now $ = I_n + \sum_i(\alpha_i - 1)\mathbf{d}_i\mathbf{d}_i^\top$ with $\alpha_i > 0$ and the $\mathbf{d}_i$'s orthonormal. Then the eigenvalues of potentially different from 1 are the α_i's (α_i is the eigenvalue associated with $\mathbf{d}_i$), so that

$$
\log|| = \sum_i \log\alpha_i.
$$

Moreover, we have $^{-1} = I_n - \sum_i \beta_i\mathbf{d}_i\mathbf{d}_i^\top$ with $\beta_i = 1 - 1/\alpha_i$ and so

$$
\mathrm{tr}(^{-1*}) = \mathrm{tr}(^*) - \sum_i \beta_i\mathbf{d}_i^{\top *}\mathbf{d}_i.
$$

Gathering the previous relation, we finally obtain the desired result.

Proof of Theorem 3.1

From Equation 15 we see that the only dependency of $D(g^*, g_{\mathbf{m},})$ in $\mathbf{m}$ is in the quadratic term $(\mathbf{m}-\mathbf{m}^*)^{\top -1}(\mathbf{m}-\mathbf{m}^*)$. As is definite positive, this term is ≥ 0, and so it is minimized for $\mathbf{m} = \mathbf{m}^*$. Next, we see that the derivative in α_i is given by (here and in the sequel, we see $D(g^*, g_{\mathbf{m},})$ as a function of $\mathbf{v} = (\alpha_i)_i$ and $\mathbf{d} = (\mathbf{d}_i)_i$)

$$\frac{\partial D}{\partial \alpha_i}(\mathbf{v}, \mathbf{d}) = \frac{1}{\alpha_i} - \frac{1}{\alpha_i^2}\mathbf{d}_i^{\top *}\mathbf{d}_i = \frac{1}{\alpha_i^2}\left(\alpha_i - \mathbf{d}_i^{\top *}\mathbf{d}_i\right).$$

Thus, for fixed $\mathbf{d}$, D is decreasing in α_i for $\alpha_i < \mathbf{d}_i^{\top *}\mathbf{d}_i$ and then increasing for $\alpha_i > \mathbf{d}_i^{\top *}\mathbf{d}_i$, which shows that, for fixed $\mathbf{d}$, it is minimized for $\alpha_i = \mathbf{d}_i^{\top *}\mathbf{d}_i$. For this value (and $\mathbf{m} = \mathbf{m}^*$) we have

$$D(g^*, g_{\mathbf{m}^*,}) = \sum_{i=1}^{k}\left[\log(\mathbf{d}_i^{\top *}\mathbf{d}_i) + 1 - \mathbf{d}_i^{\top *}\mathbf{d}_i\right] + C = -\sum_{i=1}^{k}\ell(\mathbf{d}_i^{\top *}\mathbf{d}_i) + C \tag{16}$$

with $C = -\frac{1}{2}\|\mathbf{m}^*\|^2 - \log\mathscr{E} + \mathbb{E}_{g^*}(\log\phi(\mathbf{X}))$ independent from the $\mathbf{d}_i$'s. Since ℓ is decreasing and then increasing, it is clear from this expression that in order to minimize D, one must choose the $\mathbf{d}_i$'s in order to either maximize or minimize $\mathbf{d}_i^{\top *}\mathbf{d}_i$, whichever maximizes ℓ. Since the variational characterization of eigenvalues shows that eigenvectors precisely solve this problem, we get the desired result.

Proof of Theorem 3.2

In Equation 15, the $\mathbf{m}^*$ and the * that appear in the right-hand side are the mean and variance of the density g^* considered in the first argument of the Kullback–Leibler divergence. In particular, if we apply Equation 15 with $\phi \equiv 1$, we have $g^* = f$, and the $\mathbf{m}^*$ and * of the right-hand side become 0 and I_n, respectively, so that

$$D(f, g_{\mathbf{m},}) = \frac{1}{2}\sum_i\left(\log\alpha_i - \left(1 - \frac{1}{\alpha_i}\right)\right) + \frac{1}{2}\mathbf{m}^{\top -1}\mathbf{m}.$$

Now, if we consider $\mathbf{m} = \mathbf{m}^*$ and $= I + (\alpha - 1)\mathbf{d}\mathbf{d}^\top$, we obtain (using $^{-1} = I - (1 - 1/\alpha)\mathbf{d}\mathbf{d}^\top$ as mentioned in the proof of Lemma 6.1)

$$D(f, g_{\mathbf{m}^*,}) = \frac{1}{2}\left(\log\alpha - \left(1 - \frac{1}{\alpha}\right)\left(1 + (\mathbf{d}^\top\mathbf{m}^*)^2\right)\right) + \frac{1}{2}\|\mathbf{m}^*\|^2.$$

Then the function $x \mapsto \log x + (1/x - 1)\gamma$ is minimized for $x = \gamma$ where it takes the value $-\ell(\gamma)$: $D(f, g_{\mathbf{m}^*,})$ is therefore minimized for $\alpha = 1 + (\mathbf{d}^\top\mathbf{m}^*)^2$ and for this value, we have

$$D(f, g_{\mathbf{m}^*,}) = -\frac{1}{2}\ell(1 + (\mathbf{d}^\top\mathbf{m}^*)^2) + \frac{1}{2}\|\mathbf{m}^*\|^2.$$

As ℓ is increasing in $[1, \infty)$, this last quantity is minimized by maximizing $(\mathbf{d}^\top\mathbf{m}^*)^2$, which is obtained for $\mathbf{d} = \mathbf{m}^*/\|\mathbf{m}^*\|$. The result is proved.

Appendix B: Choice of the auxiliary density $g^{'}$ for the von Mises–Fisher–Nakagami model

Von Mises–Fisher–Nakagami (vMFN) distributions were proposed in (Papaioannou, Geyer, and Straub 2019) as an alternative to the Gaussian parametric family to perform IS for high dimensional probability estimation. A random vector $\mathbf{X}$ drawn according to the vMFN distribution can be written as $\mathbf{X} = R\mathbf{A}$ where $\mathbf{A} = \frac{\mathbf{X}}{\|\mathbf{X}\|}$ is a unit random vector following the von Mises–Fisher distribution, and $R = \|\mathbf{X}\|$ is a positive random variable with a Nakagami distribution; further, R and $\mathbf{A}$ are independent. The vMFN pdf can be written as

$$g_{\text{vMFN}}(\mathbf{x}) = g_{\text{N}}(\|\mathbf{x}\|, p, \omega) \times g_{\text{vMF}}\left(\frac{\mathbf{x}}{\|\mathbf{x}\|}, \boldsymbol{\mu}, \kappa\right). \tag{17}$$

The density $g_{\text{N}}(\|\mathbf{x}\|, p, \omega)$ is the Nakagami distribution with shape parameter $p \geq 0.5$ and a spread parameter $\omega > 0$ defined by

$$g_{\text{N}}(\|\mathbf{x}\|, p, \omega) = \frac{2p^p}{\Gamma(p)\omega^p}\|\mathbf{x}\|^{2p-1}\exp\left(-\frac{p}{\omega}\|\mathbf{x}\|^2\right)$$

and the density $g_{\text{vMF}}(\frac{\mathbf{x}}{\|\mathbf{x}\|}, \boldsymbol{\mu}, \kappa)$ is the von Mises–Fisher distribution, given by

$$g_{\text{vMF}}\left(\frac{\mathbf{x}}{\|\mathbf{x}\|}, \boldsymbol{\mu}, \kappa\right) = C_n(\kappa)\exp\left(\kappa\boldsymbol{\mu}^T\frac{\mathbf{x}}{\|\mathbf{x}\|}\right),$$

where $C_n(\kappa)$ is a normalizing constant, $\boldsymbol{\mu}$ is a mean direction (with $\|\boldsymbol{\mu}\| = 1$) and $\kappa > 0$ is a concentration parameter.

Choosing a vMFN distribution therefore amounts to choosing the parameters $p, \omega, \boldsymbol{\mu}$, and κ. There are therefore $n + 3$ parameters to estimate, which is a significant reduction compared to the $\frac{n(n+3)}{2}$ required parameters of the Gaussian model with full covariance matrix.

Following (Papaioannou, Geyer, and Straub 2019), given a sample $\mathbf{X}_1^*, \ldots, \mathbf{X}_M^*$ distributed from g^*, the parameters ω, p, $\boldsymbol{\mu}$ and κ are set in the following way in order to define g':

$$\widehat{\omega} = \frac{1}{M}\sum_{i=1}^{M}\|\mathbf{X}_i^*\|^2 \quad \text{and} \quad \hat{p} = \frac{\widehat{\omega}^2}{\hat{\tau} - \widehat{\omega}^2} \text{ with } \hat{\tau} = \frac{1}{M}\sum_{i=1}^{M}\|\mathbf{X}_i^*\|^4$$

and

$$\hat{\boldsymbol{\mu}} = \frac{\sum_{i=1}^{M}\frac{\mathbf{X}_i^*}{\|\mathbf{X}_i^*\|}}{\|\sum_{i=1}^{M}\frac{\mathbf{X}_i^*}{\|\mathbf{X}_i^*\|}\|} \quad \text{and} \quad \hat{\kappa} = \frac{n\hat{\chi} - \hat{\chi}^3}{1 - \hat{\chi}^2} \text{ with } \hat{\chi} = \min\left(\left\|\frac{1}{M}\sum_{i=1}^{M}\frac{\mathbf{X}_i^*}{\|\mathbf{X}_i^*\|}\right\|, 0.95\right).$$

Appendix C: MCMC sampling

We consider again the test case 1 of Section 5.1 but the samples of g^* are no more generated with rejection sampling but with the Metropolis–Hastings Algorithm. The computationnal cost to generate the samples of g^* is thus much lower with MCMC but the resulting samples are dependent. The number M of g^* samples has been increased to 1000 so that the Markov chain has enough step to reach convergence. The simulation results are available in Table 7 and leads to the same conclusion as with rejection sampling.

```
##############################################################################
# Table 2. Numerical comparison on test case 1
##############################################################################

n=100         # dimension
phi=Somme

def mypi(X):
    n=np.shape(X)[1]
    f0=sp.stats.multivariate_normal.pdf(X,mean=np.zeros(n),cov=np.eye(n))
    return((phi(X)>0)*f0)
E=sp.stats.norm.cdf(-3)
N=2000
M=1000
B=500   # number of runs

Eopt=np.zeros(B)
EIS=np.zeros(B)
Eprj=np.zeros(B)
Eprm=np.zeros(B)
Eprjst=np.zeros(B)
Eprmst=np.zeros(B)
Evmfn=np.zeros(B)

SI=[]
SIP=[]
SIPst=[]
SIM=[]
SIMst=[]

# Mstar
alpha=np.exp(-3**2/2)/(E*np.sqrt(2*np.pi))
Mstar=alpha*np.ones(d)/np.sqrt(d)
# Sigmastar
vstar=3*alpha-alpha**2+1
Sigstar= (vstar-1)*np.ones((d,d))/d+np.eye(d)

Eigst=np.linalg.eigh(Sigstar)
logeigst=np.sort(np.log(Eigst[0])-Eigst[0])
deltast=np.zeros(len(logeigst)-1)

for i in range(len(logeigst)-1):
    deltast[i]=abs(logeigst[i]-logeigst[i+1])

## choice of the number of dimension
k_st=np.argmax(deltast)+1

indist=[]
for i in range(k_st):
```

```
    indist.append(np.where(np.log(Eigst[0])-Eigst[0]==logeigst[i])[0][0])

P1st=np.array(Eigst[1][:,indist[0]],ndmin=2).T
for i in range(1,k_st):
    P1st=np.concatenate((P1st,np.array(Eigst[1][:,indist[i]],ndmin=2).T)\
                        ,axis=1)    # matrix of influential directions

#np.random.seed(0)
for i in range(B):
############################## Estimation of the matrices

   ## g*-sample of size M with Metropolis-Hastings
    X=np.ones((1,n))*3/np.sqrt(n)
    param_agit=0.5
    VA0=sp.stats.multivariate_normal(mean=np.zeros(n),cov=np.eye(n))
    j=0
    while j<M:
        j=j+1
        P=VA0.rvs(size=1)
        X2=(X[-1,:]+param_agit*P)/np.sqrt(1+param_agit*param_agit)
        if(phi([X2])>0):
            X=np.append(X,[X2],axis=0)
        else:
            X=np.append(X,[X[-1,:]],axis=0)


    R=np.sqrt(np.sum(X**2,axis=1))
    Xu=(X.T/R).T

   ## estimated gaussian mean and covariance
    mm=np.mean(X,axis=0)
    Xc=(X-mm).T
    sigma =Xc @ Xc.T/np.shape(Xc)[1]
    SI.append(sigma)

   ## von Mises Fisher parameters
    normu=np.sqrt(np.mean(Xu,axis=0).dot(np.mean(Xu,axis=0).T))
    mu=np.mean(Xu,axis=0)/normu
    mu=np.array(mu,ndmin=2)
    chi=min(normu,0.95)
    kappa=(chi*n-chi**3)/(1-chi**2)

   ## Nakagami parameters
    omega=np.mean(R**2)
    tau4=np.mean(R**4)
    pp=omega**2/(tau4-omega**2)

   ###
    Eig=np.linalg.eigh(sigma)
```

```
    logeig=np.sort(np.log(Eig[0])-Eig[0])
    delta=np.zeros(len(logeig)-1)
    for j in range(len(logeig)-1):
        delta[j]=abs(logeig[j]-logeig[j+1])

    k=np.argmax(delta)+1

    indi=[]
    for l in range(k):
        indi.append(np.where(np.log(Eig[0])-Eig[0]==logeig[l])[0][0])

    P1=np.array(Eig[1][:,indi[0]],ndmin=2).T
    for l in range(1,k):
        P1=np.concatenate((P1,np.array(Eig[1][:,indi[l]],ndmin=2).T)\
                          ,axis=1)

    diagsi=np.diag(Eig[0][indi])
    sig_opt_d=P1.dot((diagsi-np.eye(k))).dot(P1.T)+np.eye(n)
    SIP.append(sig_opt_d)

   ###
    diagsist=P1st.T.dot(sigma).dot(P1st)
    sig_opt=P1st.dot(diagsist-np.eye(k_st)).dot(P1st.T)+np.eye(n)
    SIPst.append(sig_opt)

   ###
    Norm_mm=np.linalg.norm(mm)
    normalised_mm=np.array(mm,ndmin=2).T/Norm_mm
    vhat=normalised_mm.T.dot(sigma).dot(normalised_mm)
    sig_mean_d=(vhat-1)*normalised_mm.dot(normalised_mm.T)+np.eye(n)
    SIM.append(sig_mean_d)

############################################## Estimation of the integral
   ###
    Xop=sp.stats.multivariate_normal.rvs(mean=mm, cov=Sigstar,size=N)
    wop=mypi(Xop)/sp.stats.multivariate_normal.pdf(Xop,mean=mm, cov=Sigstar)
    Eopt[i]=np.mean(wop)

   ###
    Xis=sp.stats.multivariate_normal.rvs(mean=mm, cov=sigma,size=N)
    wis=mypi(Xis)/sp.stats.multivariate_normal.pdf(Xis,mean=mm, cov=sigma)
    EIS[i]=np.mean(wis)

   ###
    Xpr=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt_d,size=N)
    wpr=mypi(Xpr)/sp.stats.multivariate_normal.pdf(Xpr,mean=mm,\
                                                   cov=sig_opt_d)
    Eprj[i]=np.mean(wpr)
```

```
    ###
     Xpm=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_mean_d,size=N)
     wpm=mypi(Xpm)/sp.stats.multivariate_normal.pdf(Xpm,mean=mm,\
                                                    cov=sig_mean_d)
     Eprm[i]=np.mean(wpm)

    ###
     Xprst=sp.stats.multivariate_normal.rvs(mean=mm, cov=sig_opt,size=N)
     wprst=mypi(Xprst)/sp.stats.multivariate_normal.pdf(Xprst,mean=mm, \
                                                        cov=sig_opt)
     Eprjst[i]=np.mean(wprst)

    ###
     Xvmfn = vMFNM_sample(mu, kappa, omega, pp, 1, N)
     Rvn=np.sqrt(np.sum(Xvmfn**2,axis=1))
     Xvnu=Xvmfn.T/Rvn

     h_log=vMF_logpdf(Xvnu,mu.T,kappa)+nakagami_logpdf(Rvn,pp,omega)
     A = np.log(n) + np.log(np.pi ** (n / 2)) - sp.special.gammaln(n / 2 + 1)
     f_u = -A
     f_chi = (np.log(2) * (1 - n / 2) + np.log(Rvn) * (n - 1) - 0.5 * \
              Rvn ** 2 - sp.special.gammaln(n / 2))
     f_log = f_u + f_chi
     W_log = f_log - h_log

     wvmfn=(phi(Xvmfn)>0)*np.exp(W_log)
     Evmfn[i]=np.mean(wvmfn)

### KL divergences
dkli=np.zeros(B)
dklp=np.zeros(B)
dklm=np.zeros(B)
dklpst=np.zeros(B)

for i in range(B):
    dkli[i]=np.log(np.linalg.det(SI[i]))+sum(np.diag(Sigstar.dot\
                                            (np.linalg.inv(SI[i]))))
    dklp[i]=np.log(np.linalg.det(SIP[i]))+sum(np.diag(Sigstar.dot\
                                            (np.linalg.inv(SIP[i]))))
    dklm[i]=np.log(np.linalg.det(SIM[i]))+sum(np.diag(Sigstar.dot\
                                            (np.linalg.inv(SIM[i]))))
    dklpst[i]=np.log(np.linalg.det(SIPst[i]))+sum(np.diag(Sigstar.dot\
                                            (np.linalg.inv(SIPst[i]))))

Tabresult=np.zeros((3,7)) # table of results

Tabresult[0,0]=np.log(np.linalg.det(Sigstar))+n
Tabresult[0,1]=np.mean(dkli)
Tabresult[0,2]=np.mean(dklpst)
```

```
Tabresult[0,3]=np.mean(dklpst)
Tabresult[0,4]=np.mean(dklp)
Tabresult[0,5]=np.mean(dklm)
Tabresult[0,6]=None

Tabresult[1,0]=np.mean(Eopt-E)/E*100
Tabresult[1,1]=np.mean(EIS-E)/E*100
Tabresult[1,2]=np.mean(Eprjst-E)/E*100
Tabresult[1,3]=np.mean(Eprjst-E)/E*100
Tabresult[1,4]=np.mean(Eprj-E)/E*100
Tabresult[1,5]=np.mean(Eprm-E)/E*100
Tabresult[1,6]=np.mean(Evmfn-E)/E*100

Tabresult[2,0]=np.sqrt(np.mean((Eopt-E)**2))/E*100
Tabresult[2,1]=np.sqrt(np.mean((EIS-E)**2))/E*100
Tabresult[2,2]=np.sqrt(np.mean((Eprjst-E)**2))/E*100
Tabresult[2,3]=np.sqrt(np.mean((Eprjst-E)**2))/E*100
Tabresult[2,4]=np.sqrt(np.mean((Eprj-E)**2))/E*100
Tabresult[2,5]=np.sqrt(np.mean((Eprm-E)**2))/E*100
Tabresult[2,6]=np.sqrt(np.mean((Evmfn-E)**2))/E*100

Tabresult=np.round(Tabresult,1)

table=[["D'",Tabresult[0,0],Tabresult[0,1],Tabresult[0,2],Tabresult[0,3],
        Tabresult[0,4],Tabresult[0,5],"/"],
      ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],
       Tabresult[1,2],Tabresult[1,3],Tabresult[1,4],Tabresult[1,5],Tabresult[1,6]],
    ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],
     Tabresult[2,2],Tabresult[2,3],Tabresult[2,4],Tabresult[2,5],Tabresult[2,6]]]

Markdown(tabulate(
  table,
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}
           "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
           "$\widehat{\mathbf{\Sigma}}^{+d}_{mean}$", "vMFN"],
    tablefmt="pipe"))
```

```
<>:222: SyntaxWarning: invalid escape sequence '\%'
<>:224: SyntaxWarning: invalid escape sequence '\%'
<>:229: SyntaxWarning: invalid escape sequence '\m'
<>:229: SyntaxWarning: invalid escape sequence '\w'
<>:229: SyntaxWarning: invalid escape sequence '\w'
<>:230: SyntaxWarning: invalid escape sequence '\w'
<>:230: SyntaxWarning: invalid escape sequence '\w'
<>:231: SyntaxWarning: invalid escape sequence '\w'
<>:222: SyntaxWarning: invalid escape sequence '\%'
<>:224: SyntaxWarning: invalid escape sequence '\%'
<>:229: SyntaxWarning: invalid escape sequence '\m'
<>:229: SyntaxWarning: invalid escape sequence '\w'
<>:229: SyntaxWarning: invalid escape sequence '\w'
```

```
<>:230: SyntaxWarning: invalid escape sequence '\w'
<>:230: SyntaxWarning: invalid escape sequence '\w'
<>:231: SyntaxWarning: invalid escape sequence '\w'
/tmp/ipykernel_7763/1172887150.py:222: SyntaxWarning: invalid escape sequence '\%'
  ["Relative error (\%)",Tabresult[1,0],Tabresult[1,1],
/tmp/ipykernel_7763/1172887150.py:224: SyntaxWarning: invalid escape sequence '\%'
  ["Coefficient of variation (\%)",Tabresult[2,0],Tabresult[2,1],
/tmp/ipykernel_7763/1172887150.py:229: SyntaxWarning: invalid escape sequence '\m'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/1172887150.py:229: SyntaxWarning: invalid escape sequence '\w'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/1172887150.py:229: SyntaxWarning: invalid escape sequence '\w'
  headers=["","$\mathbf{\Sigma}^*$", "$\widehat{\mathbf{\Sigma}}^*$", "$\widehat{\mathbf{\Sigma}}_
/tmp/ipykernel_7763/1172887150.py:230: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
/tmp/ipykernel_7763/1172887150.py:230: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}_{mean}$", "${\widehat{\mathbf{\Sigma}}^{+d}_{opt}}$",\
/tmp/ipykernel_7763/1172887150.py:231: SyntaxWarning: invalid escape sequence '\w'
  "$\widehat{\mathbf{\Sigma}}^{+d}_{mean}$", "vMFN"],
```

Table 7: Numerical comparison of the estimation of $\mathscr{E} \approx 1.35 \cdot 10^{-3}$ considering the Gaussian model with the six covariance matrices defined in Section 4.2 and the vFMN model, when $\phi = \mathbb{I}_{\varphi \geq 0}$ with φ the linear function given by Equation 10. As explained in the text, the sample of g^* is generated with MCMC instead of rejection sampling. The computational cost is $N = 2000$.

	Σ^*	$\widehat{\Sigma}^*$	$\widehat{\Sigma}_{opt}$	$\widehat{\Sigma}_{mean}$	$\widehat{\Sigma}^{+d}_{opt}$	$\widehat{\Sigma}^{+d}_{mean}$	vMFN
D’	97.3	445.7	97.4	97.4	100.6	98.6	/
Relative error (%)	-0.1	-100	-0.5	-0.5	-2.9	-0.8	2.2
Coefficient of variation (%)	19.3	100	28.9	28.9	46.5	125.6	41.3

References

Agapiou, Sergios, Omiros Papaspiliopoulos, Daniel Sanz-Alonso, and Andrew M Stuart. 2017. “Importance Sampling : Intrinsic Dimension and Computational Cost.” *Statistical Science* 32 (3): 405–31. https://doi.org/10.1214/17-STS611.

Ashurbekova, Karina, Antoine Usseglio-Carleve, Florence Forbes, and Sophie Achard. 2020. “Optimal Shrinkage for Robust Covariance Matrix Estimators in a Small Sample Size Setting.”

Au, S. K., and J. L. Beck. 2003. “Important Sampling in High Dimensions.” *Structural Safety* 25 (2): 139–63. https://doi.org/10.1016/S0167-4730(02)00047-4.

Bassamboo, Achal, Sandeep Juneja, and Assaf Zeevi. 2008. “Portfolio Credit Risk with Extremal Dependence: Asymptotic Analysis and Efficient Simulation.” *Operations Research* 56 (3): 593–606. https://doi.org/10.1287/opre.1080.0513.

Benaych-Georges, Florent, and Raj Rao Nadakuditi. 2011. “The Eigenvalues and Eigenvectors of Finite, Low Rank Perturbations of Large Random Matrices.” *Advances in Mathematics* 227 (1): 494–521. https://doi.org/10.1016/j.aim.2011.02.007.

Bengtsson, Thomas, Peter Bickel, and Bo Li. 2008. "Curse-of-Dimensionality Revisited: Collapse of the Particle Filter in Very Large Scale Systems." In *Institute of Mathematical Statistics Collections*, 316–34. Beachwood, Ohio, USA: Institute of Mathematical Statistics. https://doi.org/10.1214/193940307000000518.

Bucklew, James. 2013. "Introduction to Rare Event Simulation." In, 58–61. Springer Science & Business Media. https://doi.org/10.1007/978-1-4757-4078-3.

Bugallo, Monica F., Victor Elvira, Luca Martino, David Luengo, Joaquin Miguez, and Petar M. Djuric. 2017. "Adaptive Importance Sampling: The Past, the Present, and the Future." *IEEE Signal Processing Magazine* 34 (4): 60–79. https://doi.org/10.1109/MSP.2017.2699226.

Cappé, Olivier, Randal Douc, Arnaud Guillin, Jean-Michel Marin, and Christian P. Robert. 2008. "Adaptive Importance Sampling in General Mixture Classes." *Statistics and Computing* 18 (4): 447–59. https://doi.org/10.1007/s11222-008-9059-x.

Chan, Joshua C. C., and Dirk P. Kroese. 2012. "Improved Cross-Entropy Method for Estimation." *Statistics and Computing* 22 (5): 1031–40. https://doi.org/10.1007/s11222-011-9275-7.

Chatterjee, Sourav, and Persi Diaconis. 2018. "The Sample Size Required in Importance Sampling." *The Annals of Applied Probability* 28 (2): 1099–1135. https://doi.org/10.1214/17-AAP1326.

Cornuet, Jean-Marie, Jean-Michel Marin, Antonietta Mira, and Christian P. Robert. 2012. "Adaptive Multiple Importance Sampling." *Scandinavian Journal of Statistics* 39 (4): 798–812. https://doi.org/10.1111/j.1467-9469.2011.00756.x.

El Masri, Maxime, Jérôme Morio, and Florian Simatos. 2021. "Improvement of the Cross-Entropy Method in High Dimension for Failure Probability Estimation Through a One-Dimensional Projection Without Gradient Estimation." *Reliability Engineering & System Safety* 216: 107991. https://doi.org/10.1016/j.ress.2021.107991.

El-Laham, Yousef, Víctor Elvira, and Mónica Bugallo. 2019. "Recursive Shrinkage Covariance Learning in Adaptive Importance Sampling." In *2019 IEEE 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, 624–28. IEEE. https://doi.org/10.1109/CAMSAP45676.2019.9022450.

Elvira, Víctor, Luca Martino, David Luengo, and Mónica F. Bugallo. 2019. "Generalized Multiple Importance Sampling." *Statistical Science* 34 (1): 129–55. https://doi.org/10.1214/18-STS668.

Fan, Jianqing, Yingying Fan, and Jinchi Lv. 2008. "High Dimensional Covariance Matrix Estimation Using a Factor Model." *Journal of Econometrics* 147 (1): 186–97.

Grace, Adam W., Dirk P. Kroese, and Werner Sandmann. 2014. "Automated State-Dependent Importance Sampling for Markov Jump Processes via Sampling from the Zero-Variance Distribution." *Journal of Applied Probability* 51 (3): 741–55. https://doi.org/10.1239/jap/1409932671.

Hohenbichler, Michael, and Rüdiger Rackwitz. 1981. "Non-Normal Dependent Vectors in Structural Safety." *Journal of the Engineering Mechanics Division* 107 (6): 1227–38. https://doi.org/10.1061/JMCEA3.0002777.

Kawai, Reiichiro. 2018. "Optimizing Adaptive Importance Sampling by Stochastic Approximation." *SIAM Journal on Scientific Computing* 40 (4): A2774–2800. https://doi.org/10.1137/18M1173472.

Ledoit, Olivier, and Michael Wolf. 2004. "A Well-Conditioned Estimator for Large-Dimensional Covariance Matrices." *Journal of Multivariate Analysis* 88 (2): 365–411. https://doi.org/10.1016/S0047-259X(03)00096-4.

Liu, Pei-Ling, and Armen Der Kiureghian. 1986. "Multivariate Distribution Models with Prescribed Marginals and Covariances." *Probabilistic Engineering Mechanics* 1 (2): 105–12. https://doi.org/10.1016/0266-8920(86)90033-0.

Mestre, Xavier. 2008a. "Improved Estimation of Eigenvalues and Eigenvectors of Covariance Matrices Using Their Sample Estimates." *IEEE Transactions on Information Theory* 54 (11): 5113–29. https://doi.org/10.1109/TIT.2008.929938.

———. 2008b. "On the Asymptotic Behavior of the Sample Estimates of Eigenvalues and Eigenvectors of Covariance Matrices." *IEEE Transactions on Signal Processing* 56 (11): 5353–68. https://doi.org/

10.1109/TSP.2008.929662.

Nadakuditi, Raj Rao, and Alan Edelman. 2008. "Sample Eigenvalue Based Detection of High-Dimensional Signals in White Noise Using Relatively Few Samples." *IEEE Transactions on Signal Processing* 56 (7): 2625–38. https://doi.org/10.1109/TSP.2008.917356.

Owen, Art, and Yi Zhou. 2000. "Safe and Effective Importance Sampling." *Journal of the American Statistical Association* 95 (449): 135–43. https://doi.org/10.1080/01621459.2000.10473909.

Papaioannou, Iason, Sebastian Geyer, and Daniel Straub. 2019. "Improved Cross Entropy-Based Importance Sampling with a Flexible Mixture Model." *Reliability Engineering & System Safety* 191 (November): 106564. https://doi.org/10.1016/j.ress.2019.106564.

Rubinstein, Reuven Y., and Dirk P Kroese. 2011a. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. New York; London: Springer. https://doi.org/10.1007/978-1-4757-4321-0.

———. 2011b. "The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning." In, 67–72. New York; London: Springer. https://doi.org/10.1007/978-1-4757-4321-0.

Rubinstein, Reuven Y., and Dirk P. Kroese. 2017b. *Simulation and the Monte Carlo Method*. Third edition. Wiley Series in Probability and Statistics. Hoboken, New Jersey: Wiley. https://doi.org/10.1002/9781118631980.

———. 2017a. "Simulation and the Monte Carlo Method." In, Third edition, 149–58. Wiley Series in Probability and Statistics. Hoboken, New Jersey: Wiley. https://doi.org/10.1002/9781118631980.

Uribe, Felipe, Iason Papaioannou, Youssef M. Marzouk, and Daniel Straub. 2021. "Cross-Entropy-Based Importance Sampling with Failure-Informed Dimension Reduction for Rare Event Simulation." *SIAM/ASA Journal on Uncertainty Quantification* 9 (2): 818–47. https://doi.org/10.1137/20M1344585.