Probabilistic Learning on Manifolds: an overview of the algorithm

Francisco Javier Gual Navarro
Tutor: Sanjay Govindjee
UPC supervisor: Joel Montoy Albareda

July 2020

Bachelor of Science in Civil Engineering
and
Bachelor of Science in Mathematics
at the
UNIVERSITAT POLITÈCNICA DE CATALUNYA

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de Formació Interdisciplinària Superior

Escola de Camins
Escola Tècnica Superior d'Enginyeria de Camins, Canals i Ports
UPC BARCELONATECH

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

To my Father, who passed away on March 26th, 2020.

Thanks to my advisor, Professor Sanjay Govindjee, thanks for his patience and his dedication throughout the semester.

Thanks to Roger Ghanem for his availability and his interest in our project.

Abstract

Many different algorithms to generate random data points following a particular probability distribution have been proposed.

The algorithm presented by C. Soize and R. Ghanem [1], known as *Probabilistic Learning on Manifolds* (PLoM) can also be used with constraints over the probability distribution.

Besides the initial data set given for performing more realizations of it, some constraints are given, which may be attributed to previous experiments or of physical restrictions. They can be a result of experience and knowledge of certain events.

This algorithm may have many applications, from models that need data to learn to Monte Carlo integration methods and many other numerical applications that require data to compute some calculations and reach some conclusions.

Keywords:
Statistics, constraints, principal component analysis, diffusion maps, kernel density estimation, learning.

Abstract

Hay muchos algoritmos capaces de producir nuevos puntos aleatorios que siguen cierta distribución de probabilidad aleatoria.

El algoritmo propuesto por C. Soize y R. Ghanem [1], llamado *Aprendizaje probabilistico en Variedades* (PLoM) también se puede usar con restricciones sobre la distribución de probabilidad.

Además del conjunto inicial de información, del cuál se tienen que calcular nuevas realizaciones, también se conocen algunas restricciones, que pueden venir de previos experimentos, o restricciones físicas. Pueden ser el resultado de la experiencia y del conocimiento de ciertos eventos.

Este algoritmo tiene muchas posibles aplicaciones, desde modelos que necesitan información para el aprendizaje hasta métodos de integración del tipo 'Monte Carlo' y muchas otras aplicaciones numéricas que requieren muchos datos para ciertos cálculos y llegar a determinadas conclusiones.

Palabras clave:
Estadística, restricciones, análisis de las componentes principales, mapa de difusión, estimación por núcleos de densidad, aprendizaje.

Abstract

Hi ha molts algorismes capaços de generar nous punts aleatoris que segueixen una certa distribució de probabilitat aleatòria.

L'algorisme proposat per C. Soize i R. Ghanem [1], anomenat *Aprenentatge prababilístic en Varietats* (PLoM) també es pot aplicar amb restriccions sobre la distribució de probabilitat.

Ademés del conjunt d'informació inicial, del qual s'han de calcular noves realitzacions, també es coneixen algunes restriccions, que poden venir d'experiments previs, o restriccions físiques. Poden ser fruit de l'experiència i del coneixement de certs esdeveniments.

Aquest algorisme té moltes possibles aplicacions, des de models que necessiten dades per l'aprenentatge fins a mètodes d'integració del tipus 'Monte Carlo' i moltes altres aplicacions numèriques que requereixen moltes dades per a certs càlculs i arribar a determinades conclusions.

Paraules clau:
Estadística, restriccions, anàlisi de les components principals, mapa de difusió, estimació per nuclis de densitat, aprenentatge.

AMS Classification: 65C20

Contents

1 Introduction

In this work there will be reviewed some parts of the algorithm *Probabilistic Learning on Manifolds* (PLoM) with some specific applications.

The integrity of the algorithm has been read and understood. The proofs have been checked and a few meetings have been carried out with one of the coauthors of the PLoM algorithm, Roger Ghanem, from University of Southern California (USC), to discuss some topics.

The code has been coded using both `Python` and `C`.

Some issues are presented and two applications have been computed, one without constraints and the other with constraints, and also several aspects of the results are presented.

This is a big algorithm that has to put together many mathematical aspects. And in order to converge, these several components have to fit together, thing that not always happens.

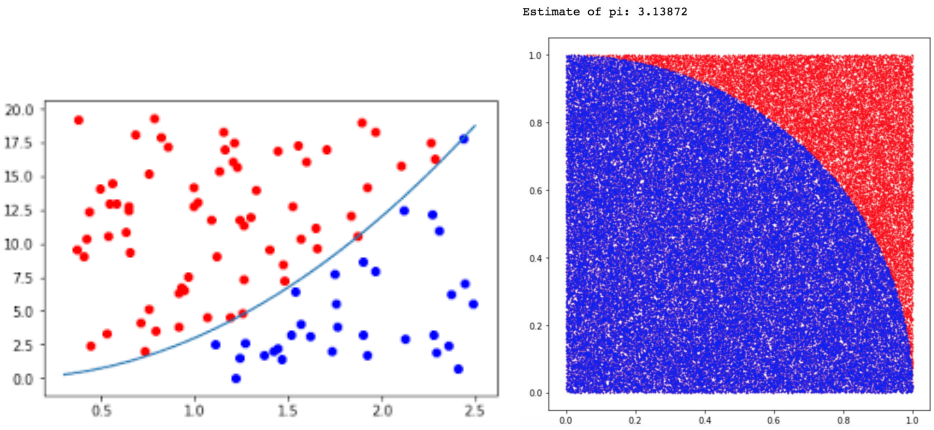

Figure 1: Monte Carlo integration and its importance of the number of realizations to compute the integrals.

Some useful notations to better follow the text.

NOTATION
Real variables are expressed with lowercase letters such as η.
Vectors are expressed with lowercase letters in bold, like $\mathbf{x}$.
Vector random variables are denoted with uppercase letters in bold, like $\mathbf{X}$.
Letters between brackets such as $[g]$ denote matrices
m_c : number of constraints. $\rightarrow \nu_c$: number of independent constraints.
n : dimension ($n = n_a + n_b$) of vectors $\mathbf{x}, \mathbf{X}$, and $\mathbf{X}^c$. $\rightarrow \nu$: dimension of $\mathbf{H}$ and $\mathbf{H}^c$.
$\mathbf{x}^j, \mathbf{x}^{c,\ell}$: realization of $\mathbf{X}, \mathbf{X}^c$.
$M = n_{\text{MC}} \times N$: number of points in the generated data set D_M^g.
N : number of points in initial data set D_N. $\rightarrow m$: number of columns in the reduced ISDE.
D_N : initial data set for $\mathbf{X}$

Table 1: Useful notations for understanding better the paper.

2 Description of the problem

Given an initial data set of points in $\mathbb{R}^n$ and some constraints, the objective is to generate more realizations of a probability distribution that fulfills the constraints and is the closest to the probability distribution of the initial data set,

$$\left\{\mathbf{x}^j, j = 1, \ldots, N\right\}, \quad \mathbf{x}^j \in \mathbb{R}^n \tag{1}$$

The initial set can be seen as samples of a random vector whose components are inputs and outputs of a certain mathematical model $\mathbf{f}$: $\mathbb{R}^{n_b} \times \mathbb{R}^{n_c} \to \mathbb{R}^{n_a}$.

$$\mathbf{A} = \mathbf{f}(\mathbf{B}, \mathbb{C}) \tag{2}$$

In which $\mathbf{B}$ is a random variable in $\mathbb{R}^{n_b}$ and it represents the components that have certain importance to determine the output of the function. Whereas $\mathbb{C}$ is also a random vector, but without statistical importance to control the model. $\mathbf{A}$ would be the random vector of the quantities of interest (QoI).

Therefore, the random vector is composed by two other vectors of interest, $\mathbf{X} = (\mathbf{A}, \mathbf{B}) \in \mathbb{R}^n$, and $n = n_a + n_b$.

The incentive for this problem is that the mapping $\mathbf{f}$ usually has a very high computational cost, thus the generation of new realizations require many calculations and an amount of time that may be out of our possibilities.

The initial data set can also be seen as data collected throughout many years, for example data collected from meteorological stations. The amount of times to gather more information may represent years or even decades, and the need for more information may not be able to wait for such a long period of time.

The need for more realizations is obvious. The more realizations of a process, the more conclusions can be taken from it. In certain applications, data, knowledge or facts and their statistics might be available, integrated from measurements, previous experiments or numerical simulations. The objective is then to generate realizations that fulfill this additional information. That is by fitting the new samples with mathematical constraints that express the available data. For example, statistics can refer to means or other moments of certain components $\mathbf{A}_k$ of $\mathbf{A}$.

Thus, it is not just enough generating more realizations of the initial data set using the PLoM, that allows for generating additional realizations of a given probabilistic density function, but it is also required that these realizations must be consistent with the constraints.

The random vector to consider, $\mathbb{X}$, may have different ranges of values for its different components. This is why random vector $\mathbb{X}$ must be scaled so all the components are in the range $[0, 1]$.

The scaling is proceeded as follows:

1. Let's consider the initial data set $\{\mathbb{x}^j, j = 1, \ldots, N\}, \quad \mathbb{x}^j \in \mathbb{R}^n$.
2. Let $\mathbb{x}^{max} \in \mathbb{R}^n$ and $\mathbb{x}^{min} \in \mathbb{R}^n$ such that $\mathbb{x}^{max}[i] = \max(\mathbb{x}^j[i])$ and $\mathbb{x}^{min}[i] = \min(\mathbb{x}^j[i]), 1 \leq j \leq N, 1 \leq i \leq n$.
3. Let $[\alpha_{\mathbf{X}}] \in \mathbb{M}_{n,n}$ be a diagonal matrix in which $[\alpha_{\mathbf{X}}]_{i,i} = (\mathbb{x}^{max}[i] - \mathbb{x}^{min}[i])$ if $\mathbb{x}^{max}[i] \neq \mathbb{x}^{min}[i]$, and $[\alpha_{\mathbf{X}}]_{i,i} = 1$ if $\mathbb{x}^{max}[i] = \mathbb{x}^{min}[i]$.

4. The scaling is such that:

$$\mathbb{X} = [\alpha_{\mathbf{X}}]\,\mathbf{X} + \mathbb{x}^{\min}, \quad \mathbf{X} = [\alpha_{\mathbf{X}}]^{-1}\left(\mathbb{X} - \mathbb{x}^{\min}\right) \tag{3}$$

and the scaled initial realizations are:

$$\mathbf{x}^j = [\alpha_{\mathbf{X}}]^{-1}\left(\mathbb{x}^j - \mathbb{x}^{\min}\right), j = 1, \ldots, N \tag{4}$$

3 Methodology of the solution

This problem is basically an optimization problem which objective is to minimize the distance between the probability distribution $P_{\mathbf{X}}(d\mathbf{x}) = p_{\mathbf{X}}(\mathbf{x})d\mathbf{x}$ of random vector $\mathbf{X}$, estimated with a Gaussian kernel-density estimation, and the probability distribution $P_{\mathbf{X}^c}(d\mathbf{x}) = p_{\mathbf{X}^c}(\mathbf{x})d\mathbf{x}$ of random vector $\mathbf{X}^c$, the solution random vector that satisfies the constraints. To find the closest pdf that satisfies the constraints, the Kullback-Leibler minimum cross entropy principle is proposed by Soize and Ghanem [1].

$$D_{KL}\left(\hat{p}; p_{\mathbf{X}}\right) = \int_{\mathbb{R}^n} \widehat{p}(\mathbf{x}) \log \frac{\hat{p}(\mathbf{x})}{p_{\mathbf{X}}(\mathbf{x})} d\mathbf{x} \tag{5}$$

The Kullback-Leibler Divergence is a modification of the formula for entropy of a probability distribution (Shannon, Equation (6)). Rather than just having one probability distribution p, the objective distribution is added.

$$H\left(p\right) = -\int_{\mathbb{R}^n} p(\mathbf{x}) \log p(\mathbf{x}) d\mathbf{x} \tag{6}$$

Essentially, what it is sought with the KL divergence is the expectation of the log difference between the probability of data in the original distribution with the new approximating distribution. Again, if it is seen in terms of log_2 it can be interpreted as "how many bits of information are expected to be lost from the original distribution to the new distribution".

The Kullback-Leibler cross entropy has some of the properties of a metric on the space of probability distribution: it's non-negative, with equality only when the two distributions are equal.

Proof: Since $log(a) \leq a - 1 \ \forall \ a > 0$

$$\begin{aligned} -D_{KL}\left(\hat{p}; p_{\mathbf{X}}\right) &= -\int_{\mathbb{R}^n} \widehat{p}(\mathbf{x}) \log \frac{\hat{p}(\mathbf{x})}{p_{\mathbf{X}}(\mathbf{x})} d\mathbf{x} \\ = \int_{\mathbb{R}^n} \widehat{p}(\mathbf{x}) \log \frac{p_{\mathbf{X}}(\mathbf{x})}{\hat{p}(\mathbf{x})} d\mathbf{x} &\leq \int_{\mathbb{R}^n} \widehat{p}(\mathbf{x}) \left(\frac{p_{\mathbf{X}}(\mathbf{x})}{\hat{p}(\mathbf{x})} - 1 \right) d\mathbf{x} \\ = \int_{\mathbb{R}^n} p_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} - \int_{\mathbb{R}^n} \widehat{p}(\mathbf{x}) d\mathbf{x} &= 1 - 1 = 0 \quad \square \end{aligned} \tag{7}$$

Unfortunately, however, it is not symmetric, and it does not follow the triangular inequality. Nevertheless, it's enough similar to a metric that it can be used to construct a sort of geometry on the space of probability distributions.

Thus, the main problem to solve is the solution to the following two equations,

$$p_{\mathbf{X}^c} = \arg \min_{\hat{p} \in \mathcal{C}_{\mathrm{ad},\mathbf{X}^c}} \int_{\mathbb{R}^n} \hat{p}(\mathbf{x}) \log \frac{\hat{p}(\mathbf{x})}{p_{\mathbf{X}}(\mathbf{x})} d\mathbf{x} \tag{8}$$

where the admissible set $\mathcal{C}_{\mathrm{ad},\mathrm{X}^c}$ is defined by,

$$C_{\mathrm{ad},\mathbf{X}^c} = \left\{ \mathrm{x} \mapsto \widehat{p}(\mathrm{x}) : \mathbb{R}^n \to \mathbb{R}^+, \int_{\mathbb{R}^n} \hat{p}(\mathbf{x}) d\mathrm{x} = 1, \int_{\mathrm{R}^n} \mathbf{g}^c(\mathbf{x}) \hat{p}(\mathbf{x}) d\mathbf{x} = \beta^c \right\} \tag{9}$$

In this way, the new realizations subject to certain constraints will be realizations of the random vector $\mathbf{X}^c$.

The analytical solution to this problem may be very difficult to calculate or it may not even be possible to compute the integrals. Even calculating the integrals with numerical methods may bring some convergence problems.

This is why an iterative algorithm is proposed. The algorithm basically consists of a Newton's Method with a Markov chain for every iteration, and some reductions to achieve a faster convergence.

The following sections talk about the algorithm, some issues that may arise and a test of the algorithm and two other practical applications and their results.

4 Summary of the algorithm

We consider a non-Gaussian random vector $\mathbf{X} \in \mathbb{R}^n$ whose unknown probability distribution has to satisfy some constraints, defined in (9). These constraints usually will come from experience and knowledge.

It is important to state that there are many kinds of constraints, for instance, the mean or other moments of some components of the vector, relations between components are also valid constraints.

The algorithm is explained in more detail in the article of Soize and Ghanem [1], they also may refer to other papers for some mathematical proofs. The steps of this algorithm can be summarized as follows:

4.1 PCA

First, a principal component analysis (PCA) of $\mathbf{X}$ of dimension $\nu \leq n$ using the initial set $\{\mathbf{x}^j, j = 1, ..., N\}$ is applied, which leads to the random vector $\mathbf{H}$ with values in $\mathbb{R}^\nu$ centered. It is normalized and it has as covariance matrix $[I_\nu]$, for which the N independent samples are $\{\boldsymbol{\eta}^j, j = 1, ..., N\}$:

$$\mathbf{H} = [\mu]^{-1/2}[\Phi]^T(\mathbf{X} - \hat{\mathbf{x}}) \tag{10}$$

In which,

- $\mathbf{H} \in \mathbb{R}^\nu$, having $1 \leq \nu \leq n$,
- $[\mu]$ is the diagonal $(\nu \times \nu)$ matrix such that $[\mu]_{\alpha\beta} = \mu_\alpha \delta_{\alpha\beta}$, μ_α are the eigenvalues of the covariance matrix of $\mathbf{X}$,
- $[\Phi] = [\varphi^1, \dots \varphi^\nu] \in \mathbb{M}_{n,\nu}$, where $[\varphi^1, \dots \varphi^\nu]$ are the orthonormal eigenvectors of the covariance matrix of $\mathbf{X}$.
- $\hat{\mathbf{x}} \in \mathbb{R}^n$ mean vector of the initial data set.

Representations of $\mathbf{H}$ are such that $\boldsymbol{\eta}^j = [\mu]^{-1/2}[\Phi]^T(\mathbf{x}^j - \hat{\mathbf{x}}) \in \mathbb{R}^v$.

4.2 KDE

Following the PCA, a Gaussian KDE (kernel-density estimation) of the probability density function $p_{\mathbf{H}}$ of random vector $\mathbf{H}$ is calculated:

$$p_{\mathbf{H}}(\eta) = c_\nu \rho(\boldsymbol{\eta}), \quad c_\nu = \frac{1}{\left(\sqrt{2\pi}\hat{s}_\nu\right)^\nu} \tag{11}$$

$$\rho(\boldsymbol{\eta}) = \frac{1}{N}\sum_{j=1}^{N} \exp\left\{-\frac{1}{2\hat{s}_\nu^2}\left\|\frac{\hat{s}_\nu}{s_\nu}\boldsymbol{\eta}^j - \boldsymbol{\eta}\right\|^2\right\} \tag{12}$$

$$s_\nu = \left\{\frac{4}{N(2+\nu)}\right\}^{1/(\nu+4)}, \quad \widehat{s}_\nu = \frac{s_\nu}{\sqrt{s_\nu^2 + (N-1)/N}} \tag{13}$$

These first reductions and approximations can be graphically seen in Figure 2.

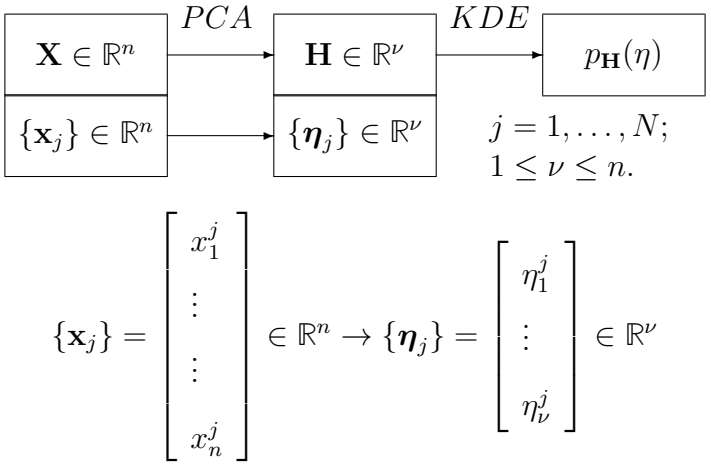

Figure 2: Graphical scheme of the first reductions.

4.3 Reformulation of the optimization problem

After the PCA and the KDE, the next step is setting again the optimization problem defined by Equations (8) and (9) for finding the probability density function $p_{\mathbf{H}^c}$ of $\mathbf{H}^c$ in terms of probability density function $p_{\mathbf{H}}$ of $\mathbf{H}$ and of the reformulated constraints in terms of $p_{\mathbf{H}^c}$:

The same procedure is applied to $\mathbf{X}^c$,

$$\mathbf{H}^c = [\mu]^{-1/2}[\Phi]^T \left(\mathbf{X}^c - \underline{\hat{\mathbf{x}}}\right), \tag{14}$$

which must be subject to the following restrictions,

$$\widetilde{\mathbf{h}}^c(\boldsymbol{\eta}) = \mathbf{g}^c\left(\underline{\widehat{\mathbf{x}}} + [\Phi][\mu]^{1/2}\boldsymbol{\eta}\right), \int_{\mathbb{R}^v} \widetilde{\mathbf{h}}^c(\eta) p_{\mathbf{H}^c}(\eta) d\eta = \beta^c. \tag{15}$$

In order that the projected constraints be algebraically independent it is proposed by Soize and Ghanem in [1], to carry out the definition of the $\mathbb{R}^{m_c}$ random variable $\widetilde{\mathbf{h}}^c(\mathbf{H})$ in a $\mathbb{R}^{\nu_c}$ random variable $\mathbf{h}^c(\mathbf{H})$.

$$\mathbf{h}^c(\boldsymbol{\eta}) = [\Psi]^T \widetilde{\mathbf{h}}^c(\boldsymbol{\eta}) = [\Psi]^T \mathbf{g}^c\left(\widehat{\mathbf{x}} + [\Phi][\mu]^{1/2}\boldsymbol{\eta}\right) \in \mathbb{R}^{\nu_c}, 1 \leq \nu_c \leq \nu. \tag{16}$$

Where $[\Psi]$ is the matrix of eigenvectors of the covariance of $\widetilde{\mathbf{h}}^c(\mathbf{H})$. In this way the problem stays as follows:

$$p_{\mathbf{H}^c} = \arg \min_{\hat{p} \in \mathcal{C}_{\text{ad},\mathbf{H}^c}} \int_{\mathbb{R}^v} \widehat{p}(\boldsymbol{\eta}) \log \frac{\widehat{p}(\boldsymbol{\eta})}{p_{\mathbf{H}}(\boldsymbol{\eta})} d\boldsymbol{\eta} \tag{17}$$

with the following restrictions:

$$\int_{\mathbb{R}^v} \mathbf{h}^c(\eta) p_{\mathbf{H}^c}(\eta) d\eta = \mathbf{b}^c (= [\Psi]^T \beta^c) \tag{18}$$

4.4 Optimization using Lagrange multipliers

Continuing with the previous section, the optimization problem with constraints is represented as an optimization problem using Lagrange multipliers (λ_0, λ) where λ_0 is associated with the normalization constraint.

The new objective random variable $\mathbf{H}^c$ is the closest to $\mathbf{H}$ and at the same time subject to the constraints in (18). And the function to minimize is the Lagrangian associated to equations (17) and (18):

$$\begin{aligned}\mathcal{L}ag\left(\hat{p};\lambda_0,\lambda\right) = &\int_{\mathbb{R}^\nu} \hat{p}(\boldsymbol{\eta}) \log \frac{\hat{p}(\boldsymbol{\eta})}{p_{\mathbf{H}}(\boldsymbol{\eta})} d\boldsymbol{\eta} \\ &+ (\lambda_0 - 1)\left(\int_{\mathbb{R}^\nu} \hat{p}(\boldsymbol{\eta}) d\boldsymbol{\eta} - 1\right) \\ &+ \left\langle \lambda, \int_{\mathbb{R}^\nu} \mathbf{h}^c(\boldsymbol{\eta})\hat{p}(\boldsymbol{\eta}) d\boldsymbol{\eta} - \mathbf{b}^c \right\rangle\end{aligned} \tag{19}$$

The solution of (19) results into,

$$\boxed{p_{\mathbf{H}^c}(\boldsymbol{\eta}) = c_\nu \exp\left\{-\psi(\boldsymbol{\eta}) - \lambda_0^{\text{sol}} - < \lambda^{\text{sol}}, \mathbf{h}^c(\boldsymbol{\eta}) >\right\}} \tag{20}$$

$$\boxed{\int_{\mathbb{R}^\nu} c_\nu \exp\left\{-\psi(\eta) - \lambda_0 - < \lambda, \mathbf{h}^c(\eta) >\right\} d\eta = 1} \tag{21}$$

$$\boxed{\int_{\mathbb{R}^\nu} c_\nu \mathbf{h}^c(\boldsymbol{\eta}) \exp\left\{-\psi(\boldsymbol{\eta}) - \lambda_0 - < \lambda, \mathbf{h}^c(\boldsymbol{\eta}) >\right\} d\boldsymbol{\eta} = \mathbf{b}^c} \tag{22}$$

in which λ is in the admissible open subset $C_{\text{ad},\lambda}$,

$$C_{\text{ad},\lambda} = \left\{\lambda \in \mathbb{R}^{\nu_c}, \int_{\mathbb{R}^\nu} e^{-\psi(\boldsymbol{\eta}) - \langle \lambda, \mathbf{h}^c(\boldsymbol{\eta})\rangle} d\boldsymbol{\eta} < +\infty\right\} \tag{23}$$

Where the pdf of random vector $\mathbf{H}$ is expressed as follows:

$$p_{\mathbf{H}}(\boldsymbol{\eta}) = c_v e^{-\psi(\boldsymbol{\eta})}, \quad \psi(\boldsymbol{\eta}) = -\log \rho(\boldsymbol{\eta}) \tag{24}$$

4.5 Reformulation introducing a random vector $\mathbf{H}_\lambda$ and reconstruction of the optimization problem

Next, the problem is reformulated again by introducing a random vector $\mathbf{H}_\lambda$ and eliminating λ_0.

The random vector $\mathbf{H}_\lambda$ is introduced,

$$p_{\mathbf{H}\lambda}(\boldsymbol{\eta}) = c_0(\lambda) \exp\left\{-\psi(\boldsymbol{\eta}) - < \lambda, \mathbf{h}^c(\boldsymbol{\eta}) >\right\} \tag{25}$$

having $c_0(\lambda)$ as the normalization constant. It can be quickly proved that $c_0(\lambda) = c_\nu \exp\{\lambda_0\}$ adn that $c_0(\lambda) = \left(\int_{\mathbb{R}^\nu} \exp\left\{-\psi(\boldsymbol{\eta}) - < \lambda, \mathbf{h}^c(\boldsymbol{\eta}) >\right\}\right)^{-1}$, and, subsequently, for all $\boldsymbol{\eta} \in \mathbb{R}^\nu$,

$$p_{\mathbf{H}^c}(\boldsymbol{\eta}) = \{p_{\mathbf{H}\lambda}(\boldsymbol{\eta})\}_{\lambda=\lambda^{\text{sol}}}, \quad p_{\mathbf{H}}(\boldsymbol{\eta}) = \{p_{\mathbf{H}\lambda}(\boldsymbol{\eta})\}_{\lambda=0} \tag{26}$$

$p_{\mathbf{H}\lambda}$ is no other thing that the solution to the Equation $\boldsymbol{\nabla}\mathcal{L}ag\left(\hat{p};\lambda_0,\lambda\right)=\mathbf{0}$.

4.6 Definition of a convex function for calculating λ^{sol}

The introduction of random variable $\mathbf{H}_\lambda$ as the solution to the Equation $\boldsymbol{\nabla}\mathcal{L}ag\left(\hat{p};\lambda_0,\lambda\right)=\mathbf{0}$ in $\mathbb{R}^\nu$ admits the construction of an algorithm that iterates over λ for computing the optimal value λ^{sol} of the Lagrange multiplier λ.

To do so, $\hat{p}$ has to be substituted for $p_{\mathbf{H}\lambda}$ in Equation (19):

$$\begin{aligned}
\mathcal{L}ag\left(p_{\mathbf{H}\lambda};\lambda_0,\lambda\right) &= \int_{\mathbb{R}^\nu} p_{\mathbf{H}\lambda}(\boldsymbol{\eta})\log\frac{p_{\mathbf{H}\lambda}(\boldsymbol{\eta})}{p_{\mathbf{H}}(\boldsymbol{\eta})}d\boldsymbol{\eta} \\
&+ (\lambda_0-1)\left(\int_{\mathbb{R}^\nu} p_{\mathbf{H}\lambda}(\boldsymbol{\eta})d\boldsymbol{\eta}-1\right) \\
&+ \left\langle\lambda,\int_{\mathbb{R}^\nu}\mathbf{h}^c(\boldsymbol{\eta})p_{\mathbf{H}\lambda}(\boldsymbol{\eta})d\boldsymbol{\eta}-\mathbf{b}^c\right\rangle \\
&= \int_{\mathbb{R}^\nu} p_{\mathbf{H}\lambda}(\boldsymbol{\eta})\log\left(c_0(\lambda)\times\exp\left[-<\lambda,\mathbf{h}^c(\boldsymbol{\eta})>\right]\right)d\boldsymbol{\eta} \\
&+ \left\langle\lambda,\int_{\mathbb{R}^\nu}\mathbf{h}^c(\boldsymbol{\eta})p_{\mathbf{H}\lambda}(\boldsymbol{\eta})d\boldsymbol{\eta}-\mathbf{b}^c\right\rangle \\
&= \log\left(c_0(\lambda)\right)-\langle\lambda,\mathbf{b}^c\rangle=\Gamma(\lambda)
\end{aligned} \tag{27}$$

thus, the function to minimize and which minimum is achieved at $\lambda=\lambda^{sol}$ is expressed as:

$$\Gamma(\lambda) = -<\lambda,\mathbf{b}^c>+\log c_0(\lambda) \tag{28}$$

with gradient,

$$\boldsymbol{\nabla}\Gamma(\lambda) = -\mathbf{b}^c+E\left\{\mathbf{h}^c\left(\mathbf{H}_\lambda\right)\right\} \tag{29}$$

and hessian matrix,

$$\left[\Gamma''(\lambda)\right] = -\left[\text{cov}\left\{\mathbf{h}^c\left(\mathbf{H}_\lambda\right)\right\}\right] \tag{30}$$

.

that can also calculated as,

$$\left[\text{cov}\left\{\mathbf{h}^c\left(\mathbf{H}_\lambda\right)\right\}\right] = E\left\{\mathbf{h}^c\left(\mathbf{H}_\lambda\right)\mathbf{h}^c\left(\mathbf{H}_\lambda\right)^T\right\}-E\left\{\mathbf{h}^c\left(\mathbf{H}_\lambda\right)\right\}E\left\{\mathbf{h}^c\left(\mathbf{H}_\lambda\right)\right\}^T \tag{31}$$

$\Gamma(\lambda)$ is, as seen before, the result of substituting $\hat{p}$ for $p_{\mathbf{H}\lambda}$ in Equation (19). And the object of iterating over λ in $\Gamma(\lambda)$ is basically optimizing the Lagrangian in this new form, that is $\Gamma(\lambda)$.

Because $-\Gamma$ is convex, this optimization problem would be convex if the set of admissible λ were convex and λ^{sol} would be the unique global minimum. However, it cannot be proven that $C_{\text{ad},\lambda}$, defined in (23), is convex for arbitrary constraints $\mathbf{h}^c$.

The iterative method will be the Newton's method applied to function $\boldsymbol{\nabla}\Gamma(\lambda)$ starting from an initial point $\lambda^1 = -\left[\Gamma''(\mathbf{0})\right]^{-1}\boldsymbol{\nabla}\Gamma(\mathbf{0})$ in $C_{\mathrm{ad},\lambda}$. This method will find λ^{sol} such that $\boldsymbol{\nabla}\Gamma(\lambda^{sol}) = \mathbf{0}$, and every iteration is calculated as,

$$\lambda^{i+1} = \lambda^i - \left[\Gamma''\left(\lambda^i\right)\right]^{-1}\boldsymbol{\nabla}\Gamma\left(\lambda^i\right) \tag{32}$$

λ^1 can be calculated using the empirical estimators (see Equation (33)) of the expectation vector and the covariance matrix of $\mathbf{h}^c(\mathbf{H})$, and in every iteration, for a given $\lambda^i \in C_{\mathrm{ad},\lambda}$, the right hand side of equations (29) and (31) can be estimated by computing realizations of $\mathbf{H}_{\lambda^i}$ using a Markov-Chain Monte-Carlo algorithm.

The empirical expectation of a given function $f : \mathbb{R}^\nu \to \mathbb{R}^{\nu_c}$ is expressed as:

$$\int_{\mathbb{R}^\nu} f(\eta) p_{\mathbf{H}^c}(\eta) d\eta = \frac{1}{n}\sum_{i=1}^{n} f\left(\eta_i\right) \tag{33}$$

The error at every iteration i is controlled calculating,

$$\mathrm{err}(i) = \frac{\left\|\mathbf{b}^c - E\left\{\mathbf{h}^c\left(\mathbf{H}_{\lambda^i}\right)\right\}\right\|}{\left\|\mathbf{b}^c\right\|} = \frac{\left\|\boldsymbol{\nabla}\Gamma\left(\lambda^i\right)\right\|}{\left\|\mathbf{b}^c\right\|} \tag{34}$$

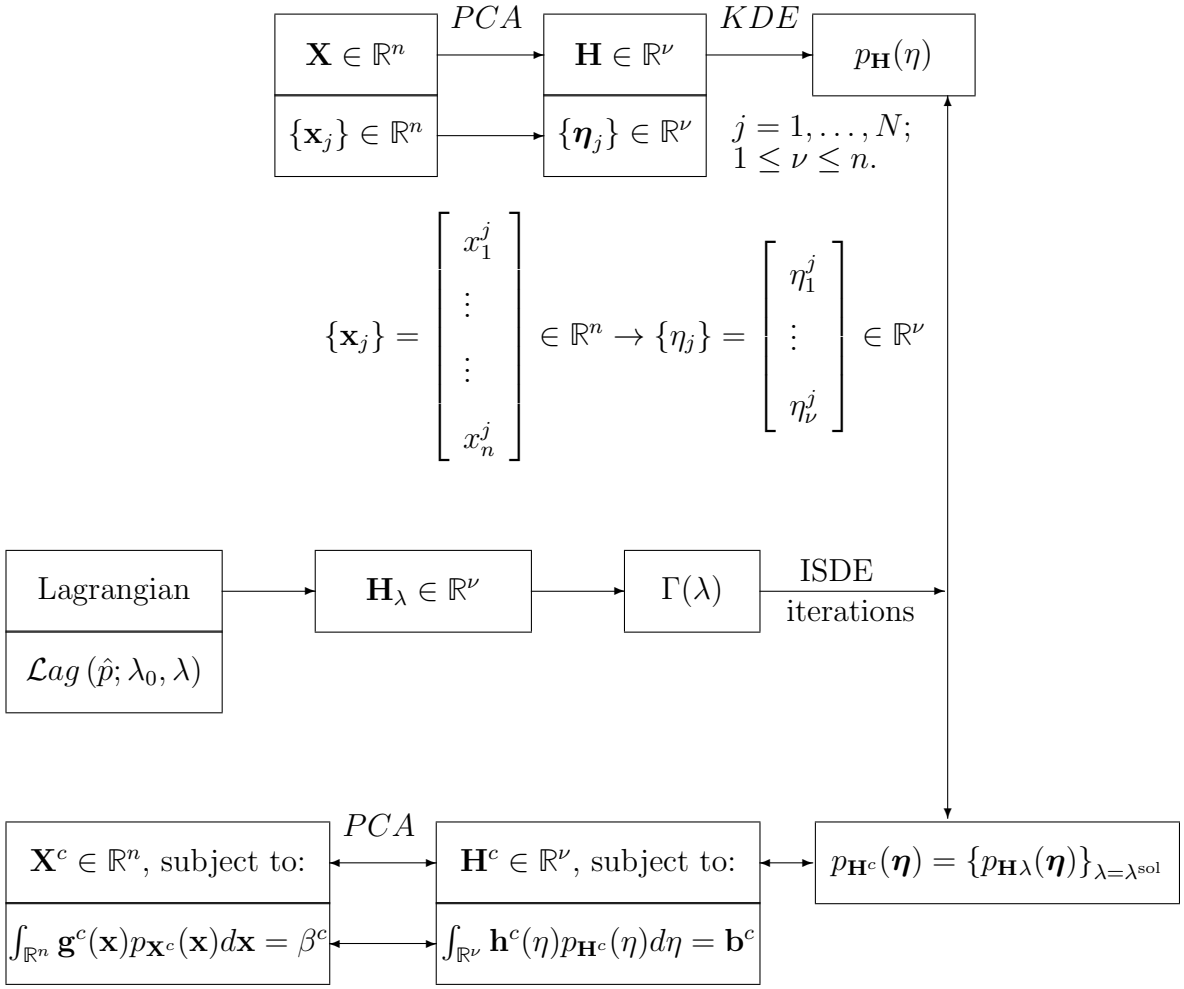

Figure 3: Big picture of the method proposed.

4.7 Nonlinear ISDE for the generator of random variable $\mathbf{H}_\lambda$

The construction of a nonlinear Ito Stochastic Differential Equation (ISDE) to generate realizations of random variable $\mathbf{H}_\lambda$ allows us to compute the empirical gradient and covariance.

The ISDE presented in page 122 of the work of Soize and Ghanem [1], is written in the following way:

$$d\left[\mathbf{U}_\lambda(t)\right] = \left[\mathbf{V}_\lambda(t)\right] dt \tag{35}$$

$$d\left[\mathbf{V}_\lambda(t)\right] = \left[L_\lambda\left(\left[\mathbf{U}_\lambda(t)\right]\right)\right] dt - \frac{1}{2} f_0 \left[\mathbf{V}_\lambda(t)\right] dt + \sqrt{f_0} d\left[\mathbf{W}_\lambda^{\text{wien}}(t)\right] \tag{36}$$

where $\mathbf{U}$ and $\mathbf{V} \in \mathbb{M}_{\nu,N}$ and with the initial condition at t = 0,

$$\left[\mathbf{U}_\lambda(0)\right] = \left[\eta^{\text{init}}\right], \quad \left[\mathbf{V}_\lambda(0)\right] = \left[\nu^{\text{init}}\right] \tag{37}$$

in which $\left\{\left(\left[\mathbf{U}_\lambda(t)\right], \left[\mathbf{V}_\lambda(t)\right]\right), t \in \mathbb{R}^+\right\}$ is a stochastic process with values in $\mathbb{M}_{\nu,N} \times \mathbb{M}_{\nu,N}$, which is composed of the independent stochastic processes $\left\{\left(\mathbf{U}_\lambda^\ell(t), \mathbf{V}_\lambda^\ell(t)\right), t \in \mathbb{R}^+\right\}$

such that $[\mathbf{U}_\lambda(t)] = \left[\mathbf{U}_\lambda^1(t) \dots \mathbf{U}_\lambda^N(t)\right]$ and $[\mathbf{V}_\lambda(t)] = \left[\mathbf{V}_\lambda^1(t) \dots \mathbf{V}_\lambda^N(t)\right]$ and where:

1. "$f_0 > 0$ is a free parameter (damping parameter) allowing the dissipation to be controlled in the stochastic dynamical system. This parameter is chosen such that $f_0 < 4$. The value, 4, corresponds to the critical damping rate of the linearized ISDE associated with Equations (35) and (36).

2. $\left\{\left[\mathbf{W}_\lambda^{\text{wien}}(t)\right], t \in \mathbb{R}^+\right\}$ is the stochastic process, defined on $(\Theta, \mathcal{T}, \mathcal{P})$, indexed by $\mathbb{R}^+$, with values in $\mathbb{M}_{\nu,N}$, for which the columns of $\left[\mathbf{W}_\lambda^{\text{wien}}(t)\right]$ are N independent copies of the $\mathbb{R}^\nu$ -valued normalized Wiener process whose matrix-valued autocorrelation function is $\min(t, t')[I_\nu]$. When coded, this random matrices are generated such that all the components are Gaussian, centered with standard deviation equal to Δt. The relation with λ is such that, if λ and λ' are two distinct values in $C_{\text{ad},\lambda}$, then the two stochastic processes $\left\{\left[\mathbf{W}_\lambda^{\text{wien}}(t)\right], t \in \mathbb{R}^+\right\}$ and $\left\{\left[\mathbf{W}_{\lambda'}^{\text{wicn}}(t)\right], t \in \mathbb{R}^+\right\}$ are independent.

3. $[u] \mapsto [L_\lambda([u])]$ is a nonlinear mapping from $\mathbb{M}_{\nu,N}$ into $\mathbb{M}_{\nu,N}$, expressed in the following form as negative of the gradient of the potential $\mathcal{V}_\lambda$

$$[L_\lambda([u])]_{\alpha\ell} = -\frac{\partial}{\partial u_\alpha^\ell}\mathcal{V}_\lambda\left(\mathbf{u}^\ell\right), \quad \alpha = 1, \dots, \nu, \quad \ell = 1, \dots, N \tag{38}$$

   in which $[u] = \left[\mathbf{u}^1 \dots \mathbf{u}^N\right] \in \mathbb{M}_{\nu,N}$ with $\mathbf{u}^\ell = \left(u_1^\ell, \dots u_\nu^\ell\right) \in \mathbb{R}^\nu$. For ℓ fixed in $\{1, \dots, N\}$, the Hamiltonian of the associated conservative homogeneous dynamical system related to stochastic process $\left\{\left(\mathbf{U}^\ell(t), \mathbf{V}^\ell(t)\right), t \in \mathbb{R}^+\right\}$ is thus written as $\mathbb{H}_\lambda\left(\mathbf{u}^\ell, \mathbf{v}^\ell\right) = \frac{1}{2}\left\|\mathbf{v}^\ell\right\|^2 + \mathcal{V}_\lambda\left(\mathbf{u}^\ell\right)$.

4. $\left[\eta^{\text{init}}\right]$ and $\left[\nu^{\text{init}}\right]$ are the matrices in $\mathbb{M}_{\nu,N}$ defined as $\left[\eta^{\text{init},\,1}\right] = \left[\boldsymbol{\eta}^1 \dots \boldsymbol{\eta}^N\right] \in \mathbb{M}_{\nu,N}$, and $\left[\nu^{\text{init},\,1}\right]$ is shuch that its N columns are independent realizations of Gaussian, centered, random variable in $\mathbb{R}^\nu$ for which its covariance matrix is $[I_\nu]$. For the following iterations, these matrices are defined as $\left[\eta^{\text{init},i}\right] = \left[\begin{array}{c} \eta_{\lambda^{i-1}}^{n_{\text{MC}}} \end{array}\right]$, $\left[\nu^{\text{init},i}\right] = \left[\begin{array}{c} \nu_{\lambda^{i-1}}^{n_{\text{MC}}} \end{array}\right]$"[1].

The presented ISDE admits a unique invariant measure, proved in the work of Soize [2],

$$\otimes_{\ell=1}^N \left\{p_{\mathbf{H}\lambda}(\mathbf{u}')\, p_{\mathbf{G}}(\mathbf{v}')\, d\mathbf{u}' d\mathbf{v}'\right\}. \tag{39}$$

The Hamiltonian of the associated dynamical system $\left\{\left(\mathbf{U}^\ell(t), \mathbf{V}^\ell(t)\right), t \in \mathbb{R}^+\right\}$ related to stochastic process is, $\mathbb{H}_\lambda\left(\mathbf{u}^\ell, \mathbf{v}^\ell\right) = \frac{1}{2}\left\|\mathbf{v}^\ell\right\|^2 + \mathcal{V}_\lambda\left(\mathbf{u}^\ell\right)$, where

$$p_{\mathbf{H}\lambda}(\boldsymbol{\eta}) = c_0(\lambda)\rho_\lambda(\boldsymbol{\eta}) \tag{40}$$

and,

$$\rho_\lambda(\boldsymbol{\eta}) = \exp\left\{-\mathcal{V}_\lambda(\boldsymbol{\eta})\right\}, \quad \mathcal{V}_\lambda(\boldsymbol{\eta}) = \psi(\boldsymbol{\eta}) + \langle\lambda, \mathbf{h}^c(\boldsymbol{\eta})\rangle \tag{41}$$

.

In this way, the realizations of $[\mathbf{H}_\lambda]$ are obtained as written,

$$[\mathbf{H}_\lambda] = \left[\mathbf{U}_\lambda^{\text{st}}(t_{\text{st}})\right] = \lim_{t \to +\infty}[\mathbf{U}_\lambda(t)] \tag{42}$$

4.8 Diffusion-maps basis

This step in the algorithm consists in projecting the nonlinear ISDE defined by Equations (35), (36) and (37) on the diffusion-maps basis. It permits conserving the dispersion of the probability distribution $p_{\mathbf{H}\lambda^i}(\boldsymbol{\eta})d\boldsymbol{\eta}$ and to prevent the dispersion of the realizations obtained with the Markov Chain Monte Carlo generator.

"The PLoM methodology has been developed for small values of N (few initial data points) for which the probability measure $p_{\mathbf{H}}(\boldsymbol{\eta})d\boldsymbol{\eta}$ is not necessarily converged. Therefore additional realizations that would be generated with this measure would not provide good realizations preserving the concentration. This is the reason why, the measure $p_{\mathbf{H}}(\boldsymbol{\eta})d\boldsymbol{\eta}$ is improved by introducing the transported probability measure $p_{\mathbf{Z}}(\boldsymbol{\eta})d\boldsymbol{z}$ of random matrix $[\mathbf{Z}_{\lambda^i}]$."[3]

The diffusion maps is a technique to reduce the dimension of the ambient space, in this case $\mathbb{R}^\nu$. An approach of the process of the diffusion maps reduction is exposed in the work of Lindenbaum et Al. [4]. The diffusion maps basis is a technique to reduce the dimension of the data space, $\mathbb{R}^N$.

The diffusion maps basis, presented in page 125 of the work of Soize and Ghanem [1], is independent on λ^i, it only depends on the initial set of points $\{\boldsymbol{\eta}^1, ..., \boldsymbol{\eta}^N\}$ and it is expressed by the matrix

$$[g] = \left[\mathbf{g}^1 \dots \mathbf{g}^m\right] \in \mathbb{M}_{N,m} \quad \text{with} \quad 1 < m \leq N \tag{43}$$

Let $[\mathbb{K}]$ be the symmetric $(N \times N)$ real matrix in which $[\mathbb{K}]_{ij} = exp\left(\frac{-1}{4\varepsilon_{diff}} \left\|\boldsymbol{\eta}^i - \boldsymbol{\eta}^j\right\|^2\right)$ that determined by the parameter $\varepsilon_{diff} > 0$, to regularize the kernel function. Let $[\mathbb{P}]$ be the transition matrix in $\mathbb{M}_N$ of a Markov chain such that $[\mathbb{P}] = [\mathbf{b}]^{-1}[\mathbb{K}]$,in which $[\mathbf{b}]$ is the matrix expressed by $[\mathbf{b}]_{ij} = \delta_{ij} \sum_{j'=1}^{N} [\mathbb{K}]_{jj'}$. It is positive-definite and diagonal.

For m fixed in $\{1, ..., N\}$, let $\mathbf{g}^1, ..., \mathbf{g}^m$ be the column eigenvectors in $\mathbb{R}^N$ of matrix $[\mathbb{P}]$ correspondent to eigenvalues sorted such that $1 = \Lambda_1 > \Lambda_2 > ... > \Lambda_m$. These eigenvectors are normalized $[g]^T[\mathbf{b}][g] = [I_m]$.

This composition depends on two factors: the dimension $m \leq N$, and the "smoothing parameter" $\varepsilon_{diff} > 0$. Most of the time,m and ε_{diff} can be chosen as presented in [1].

The new representation of the points $\{\boldsymbol{\eta}^j, j = 1, ..., N\}$ is defined representing each $\{\boldsymbol{\eta}^j\}$ by the j-th row of $[\mathbb{P}]^t[g] = [g][\Lambda]^t$, where $t > 0 \in \mathbb{Z}$ namely:

$$\Psi_t\left(\boldsymbol{\eta}_j\right): \quad \boldsymbol{\eta}_j \mapsto \left[\Lambda_1^t \mathbf{g}_j^1, \dots, \Lambda_m^t \mathbf{g}_j^m\right]^{\mathbf{T}} \in \mathbb{R}^m, \quad j \in [1, N] \tag{44}$$

where $\mathbf{g}_i^k$ denotes the i-th element of the vector $\mathbf{g}^k$.

In this way, $[\Lambda][g]^T$ columns would correspond to the reduced representation of the initial points.

The principal idea behind this representation is that the distance between two data points in the lower new representation maintains the distance of the data points in their original representation.

The diffusion maps basis $[g]$ characterizes the local geometry of $\{\boldsymbol{\eta}^j, j = 1, ..., N\}$:

$$[\mathbf{H}_{\lambda^i}] = [\mathbf{Z}_{\lambda^i}]\,[g]^T \tag{45}$$

and consequently having,

$$[\mathbf{Z}_{\lambda^i}] = [\mathbf{H}_{\lambda^i}]\,[a] \tag{46}$$

with,

$$[a] = [g] \left([g]^T [g] \right)^{-1} \in \mathbb{M}_{N,m} \tag{47}$$

In the same way, we have,

$$[\mathbf{U}_{\lambda^i}(t)] = [\mathcal{Z}_{\lambda^i}(t)]\,[g]^T, \quad [\mathbf{V}_{\lambda^i}(t)] = [\mathcal{Y}_{\lambda^i}(t)]\,[g]^T \tag{48}$$

Therefore, $[\mathbf{Z}_{\lambda^i}]$, in which $[\Lambda]$ is "hidden", can be interpreted as a linear map from $\mathbb{R}^m$ to $\mathbb{R}^\nu$. The idea of the diffusion maps is that points from $\mathbb{R}^\nu$ are projected to $\mathbb{R}^m$ in a non-linear mapping (the diffusion maps) to $[g]^T$, and this basis is assumed constant. And from this same points it is assumed that to undo the diffusion maps projection a linear map is proposed ($[\mathbf{Z}_{\lambda^i}]$), and the ISDE is solved over generating new linear maps from $\mathbb{R}^m$ to $\mathbb{R}^\nu$.

As the diffusion maps is not a bijective projection, the inverse function can not be computed. In this way this linear map as the inverse function is proposed, and the parameters ε and m are such that they reduce the mean-square error when the projection is performed and undone:

$$e_{\text{red}}(m) = \frac{\| [\text{cov}_{\text{red}}(m)] - [\text{cov}] \|_F}{\| [\text{cov}] \|_F} \tag{49}$$

where,

$$[\text{cov}] = \frac{1}{N-1} \sum_{\ell=1}^{N} \left(\boldsymbol{\eta}^\ell \right) \left(\boldsymbol{\eta}^\ell \right)^T \tag{50}$$

$$[\text{cov}_{\text{red}}(m)] = \frac{1}{N-1} \sum_{\ell=1}^{N} \left(\boldsymbol{\eta}^\ell_{\text{red}} \right) \left(\boldsymbol{\eta}^\ell_{\text{red}} \right)^T \tag{51}$$

$$[\boldsymbol{\eta}_{\text{red}}(m)] = [z_{initial}]\,[g]^T \quad , \quad [z_{initial}] = [\eta_{initial}]\,[a] \tag{52}$$

The resulting reduced ISDE concentrates the results on a manifold of $\mathbb{R}^N$ (the "data space") and it is expressed as:

$$d\,[\mathcal{Z}_{\lambda^i}(t)] = [\mathcal{Y}_{\lambda^i}(t)]\,dt \tag{53}$$

$$d\,[\mathcal{Y}_{\lambda^i}(t)] = [\mathcal{L}_{\lambda^i}\,([\mathcal{Z}_{\lambda^i}(t)])]\,dt - \frac{1}{2} f_0\,[\mathcal{Y}_{\lambda^i}(t)]\,dt + \sqrt{f_0}\,d\left[\mathcal{W}^{\text{wien}}_{\lambda^i}(t)\right] \tag{54}$$

with the projected initial conditions at $t = 0$

$$[\mathcal{Z}_{\lambda^i}(0)] = \left[n^{n_{\text{MC}}}_{\lambda^{i-1}}\right][a], \quad [\mathcal{Y}_{\lambda^i}(0)] = \left[\nu^{n_{\text{MC}}}_{\lambda^{i-1}}\right][a], \text{ a.s.} \tag{55}$$

in which random matrices $[\mathcal{L}_{\lambda^i}\,([\mathcal{Z}_{\lambda^i}(t)])]$ and $\left[\mathcal{W}^{\text{wien}}_{\lambda^i}(t)\right]$

$$[\mathcal{L}_{\lambda^i}\,([\mathcal{Z}_{\lambda^i}(t)])] = \left[L_{\lambda^i}\left([\mathcal{Z}_{\lambda^i}(t)]\,[g]^T\right)\right][a] \tag{56}$$

$$\left[\mathcal{W}^{\text{wien}}_{\lambda^i}(t)\right] = \left[\mathbf{W}^{\text{wien}}_{\lambda^i}(t)\right][a] \tag{57}$$

The realizations ($M = n_{\text{MC}} \times N$) of $[\mathbf{H}_{\lambda^i}]$ are then obtained using Equation (48):

$$\left[\eta_{\lambda^i}^{\ell}\right] = \left[z_{\lambda^i}^{\ell}\right][g]^T \in \mathbb{M}_{\nu,N}, \quad \ell = 1, \ldots, n_{\mathrm{MC}} \tag{58}$$

To solve this ISDE, a discretization of it is proposed. In this case the Störmer-Verlet algorithm, described in Appendix A, is used for finding the solution of the reduced-order ISDE defined by Equations (53) to (57). The Störmer-Verlet algorithm is explained in the work of Soize and Ghanem [1].

In the following Figure it can be seen that the reduction made by the diffusion maps basis projection is from N to m, and not a dimension reduction as the diffusion maps itself is supposed to be, explained in the work of Coifman et al. [5], but in the data space.

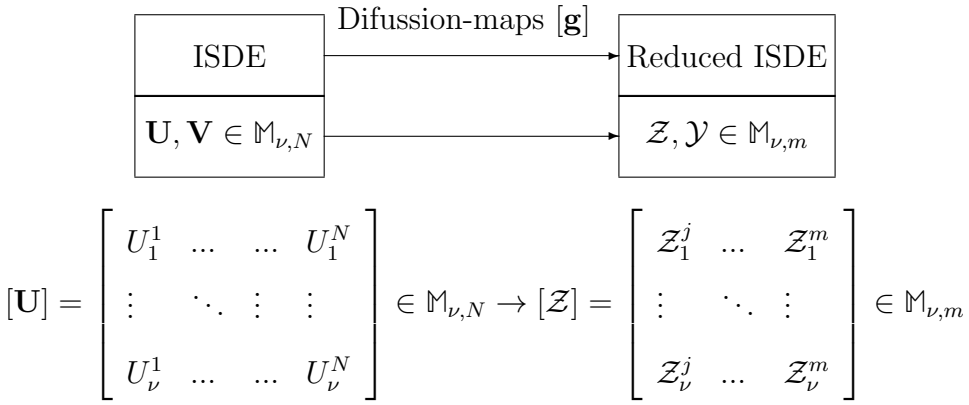

Figure 4: ISDE reduction.

4.9 Computing the additional realizations of $\mathbf{H}^c$

If the conditions of the problem are so, the constrains are physically and mathematically coherent with the initial data set, and all the steps have been computed the Newton method will iterate over λ towards the optimal point.

Once an error smaller than a given tolerance is achieved, the next step is computing the additional samples of $\mathbf{H}^c$ solving the ISDE and then concluding with additional realizations of $\mathbf{X}^c$.

At that given iteration, as the tolerance is small enough, we have, $\mathbf{H}_\lambda = \mathbf{H}^c$, and then, we undo the PCA described at Equation (10),

$$\mathbf{X}^c = \widehat{\mathbf{x}} + [\Phi][\mu]^{1/2}\mathbf{H}^c \tag{59}$$

The $M = n_{MC} \times N$ realizations $\{\mathbf{x}^{c,\ell'}, \ell' = 1, ..., M\}$ of random vector $\mathbf{X}^c$ are computed using the previous equation, having:

$$\mathbf{x}^{c,\ell'} = \widehat{\mathbf{x}} + [\Phi][\mu]^{1/2}\boldsymbol{\eta}^{c,\ell'}, \quad \ell' = 1, \ldots, M \tag{60}$$

where $\boldsymbol{\eta}^{c,\ell'}$ is the ℓ'-th column of the concatenated n_{MC} new (matrix) realizations of $[\mathbf{H}^c]$.

M usually is as big as needed. As mentioned before, computing new realizations without the PLoM algorithm may require a high computational cost and a high amount

of time. Usually the whole of the algorithm may last just a few hours, or a few days to calculate the M realizations, in comparison to the cost of the original model that can be days, weeks or even years if for example we are collecting data from weather conditions.

Finally, these new realizations can be used for estimating statistics of $\mathbf{X}^c$, adjusting mathematical models, training neural networks or other machine learning algorithms.

This method combines many parts, and several tools. Its convergence depends in how they come together and how consistent they are with the initial conditions.

5 Major issues of the algorithm

In this section the major issues of the algorithm will be presented. These are the issues faced when coding the program and analyzing its results. The PLoM with constraints has to bring together many conditions in order to converge. For example, if the constraints are incongruous with the initial data set the problem may not have solution and thus, it will not converge. Note that the set $C_{\text{ad},\lambda}$, presented in Equation (23), may not be convex or that the solution may not be in this set.

The first issue faced when running the program was the computational cost. For quite small dimensions (n) and an affordable number of initial point ($N \approx 300$) the code lasted several hours before finishing, and not getting quite the desired precision.

The second issue is related to the convergence of the algorithm, and mostly due to the diffusion maps basis [g].

5.1 Comment regarding the code

The algorithm has been programmed in Python due to the numerous matrix operations, and its advantage to work with indexes, slices,...

As `Python` is an interpreted language and the code is considerably slow to run, especially in big samples, and in order to boost the whole process, some functions have been coded in `C`. The major computational cost of the algorithm is calculating, at every step of the Markov Chain, matrix $[L_\lambda([u])]_{\alpha\ell}$. This matrix, expressed in Equation (38) is basically the gradient of the potential $\mathbb{H}_\lambda\left(\mathbf{u}^\ell, \mathbf{v}^\ell\right)$.

This matrix involves the calculation of $\boldsymbol{\nabla} \ln \rho(\boldsymbol{\eta})$ which requires the calculation of $\boldsymbol{\nabla}\rho(\boldsymbol{\eta})$ and $\rho(\boldsymbol{\eta})$, described at Equation (12). This two functions are a kernel density estimation of a data set of N points and they are basically a sum of N exponential functions and that has quite a high computational cost.

As $[L_\lambda([u])]_{\alpha\ell}$ is the concatenation of N column vectors in $\mathbb{R}^\nu$ that basically are $\boldsymbol{\nabla} \ln \rho_\lambda(\boldsymbol{\eta})$, the two KDE functions have to be calculated N times per MCMC step. That makes a cost of the order of $O(N^2)$.

Modules written in `C` are commonly used to extend the capabilities of a `Python` interpreter as well as to enable access to low-level operating system capabilities. Because `Python` is an interpreted language, certain pieces of code can be written in `C` for higher performance.

The methodology used is the following:

1. Code functions $\boldsymbol{\nabla}\rho(\boldsymbol{\eta})$ and $\rho(\boldsymbol{\eta})$ in `C`. All the variables used are `double, int and pointers`, as it is `C` and not `C++`.

2. Create a shared library from the `C` file from `Ubuntu` terminal. That is creating a "`.so`" file that can only be read from `Ubuntu`.

3. Import module `ctypes` in `Python`. It is a foreign function library for `Python`. It provides `C` compatible data types, and allows calling functions in shared libraries. It can be used to wrap these libraries in pure `Python`.

4. This module allows us to import the shared `C` library into `Python`, and have them as any other `Python` functions that we may have in any `Python` library. The functions in this shared library can be run seamlessly from `Python`.

5. Last, define what kind of inputs and outputs do the functions in the shared library take. As a compiled language, `C` requires to define the variables, and ctypes allows to define the types of variables both as inputs and as results of the functions in the shared library.

There are other ways of linking `C` and `Python` in Windows, like the module `cython`. Some of them, however, require other packages or the installation of other programs, such as `Visual Studio` or similar.

Just with this two functions in `C`, the algorithm is boosted around 100x faster in a regular size of N ($100 \leq N \leq 500$) and a small size of ν ($\nu = 10$). In Figure 5 it can be seen how faster is the module for calculating $\boldsymbol{\nabla}\rho(\boldsymbol{\eta})$ in `C` than the module only in `Python`.

Something similar happens for N ($100 \leq N \leq 500$) and a bigger size of ν ($\nu = 100$), in this case the algorithm is boosted 50 times for small N, as seen in Figure 6.

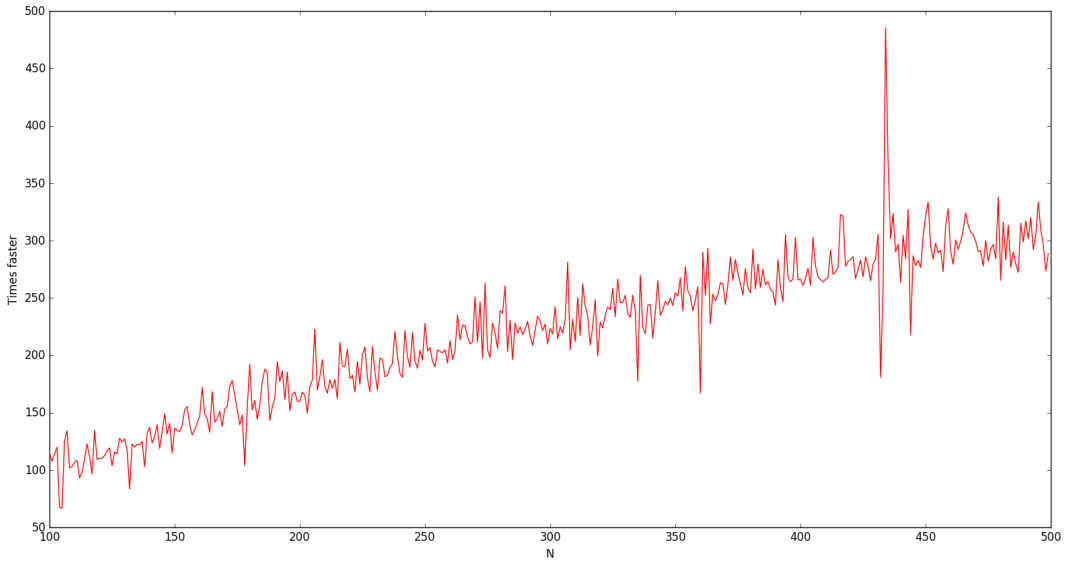

Figure 5: $\times$Times faster `C` respect to `Python` in these two functions for $\nu = 10$.

The calculation of $[L_\lambda([u])]_{\alpha\ell}$ represents 99% of the time spent in every step in the Markov Chain Monte Carlo, which means that a substantial improvement of their performance is a direct boost for the whole of the algorithm.

Tests that lasted days in running the code, this way they finish in a few hours or even minutes.

Moreover, it could also happen, specially when the function $\boldsymbol{\nabla} \ln \rho_\lambda(\boldsymbol{\eta})$ is evaluated far from the initial data $\{\boldsymbol{\eta}^j, j = 1, ..., N\}$, that some computational errors may arise. Mainly because the information storage is finite and the lowest positive real number that can be stored in a `Python` variable is $2.2250738585072014e-308$ and there may be some cases where the function $\rho(\boldsymbol{\eta})$ may take smaller values than that.

In these situations what it is proposed to do is a simplification of the gradient $\boldsymbol{\nabla} \ln \rho(\boldsymbol{\eta})$:

$$\boldsymbol{\nabla} \ln \rho(\boldsymbol{\eta}) = \frac{\left(\frac{\widehat{s}_\nu}{s_\nu}\boldsymbol{\eta}^{j^*} - \boldsymbol{\eta}\right)}{\widehat{s}_\nu^2} \tag{61}$$

where,

$$\left|\left|\frac{\widehat{s}_\nu}{s_\nu}\boldsymbol{\eta}^{j^*} - \boldsymbol{\eta}\right|\right| = \min_{1\leq j\leq N}\left|\left|\frac{\widehat{s}_\nu}{s_\nu}\boldsymbol{\eta}^{j} - \mathbf{u}^{\ell}\right|\right| \tag{62}$$

This approximation of $\boldsymbol{\nabla} \ln \rho(\boldsymbol{\eta})$ is nothing else than the limit when it is evaluated far from the initial data set. This procedure may add more computational cost to the algorithm because there is a search at every calculation of $\boldsymbol{\nabla} \ln \rho(\boldsymbol{\eta})$, but fortunately this scenario only happens in cases where $\rho(\boldsymbol{\eta})$ is very close to zero, usually far from the initial set. So if the constraints are consistent with the initial data set, *a priori*, it should not be a problem.

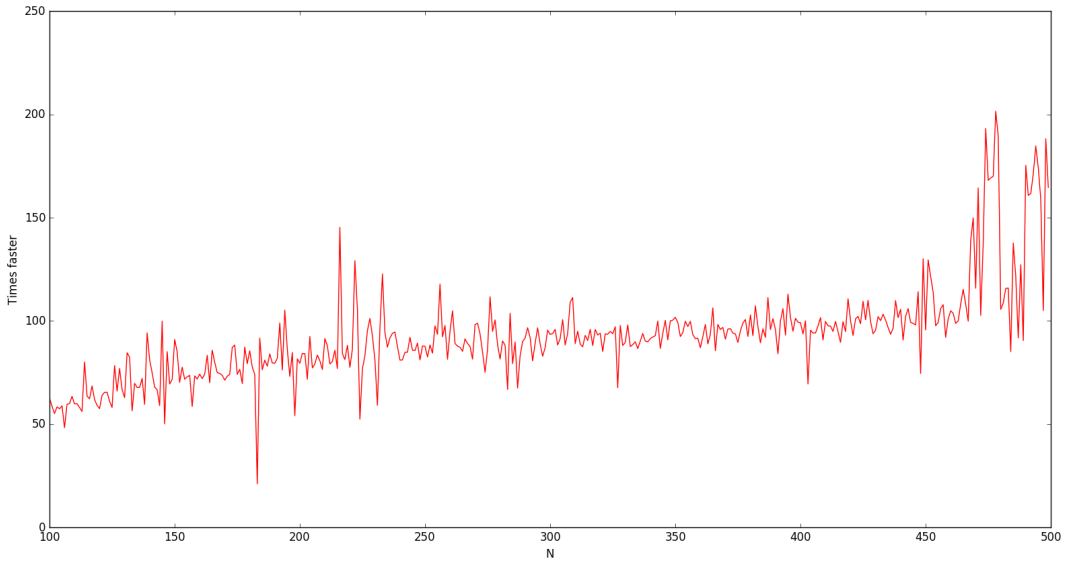

Figure 6: ×Times faster `C` respect to `Python` in these two functions for $\nu = 100$.

5.2 Convergence of the algorithm and diffusion maps accuracy

The convergence of the algorithm depends on many things, due to the fact that the whole of the algorithm puts together many different approximations and methods to solve every step.

The diffusion maps presented in section 4.8 has importance both for preserving the new realizations close and to reduce the number of needed realizations M in order to achieve more accuracy in the MCMC steps.

To analyze the convergence of the algorithm a test has been performed where the analytical solution is known. An application in Section 6 has been also computed to see the practical convergence of the Newton method. The case consists of an initial data set of realizations of a Gaussian random vector in $\mathbb{R}^2$ centered and covariance the identity and the constrains given by:

$$\int_{\mathrm{R}^n} \begin{pmatrix} x_1 \\ x_1^2 \end{pmatrix} \hat{p}(\mathbf{x}) d\mathbf{x} = \begin{pmatrix} 0.5 \\ 1.5 \end{pmatrix} \tag{63}$$

The test has been implemented without PCA and without diffusion maps, both analytically and empirically. The KDE step has been substituted by the already known probability density function, Equation (64), for the initial data set, a bivariate Gaussian centered ($\boldsymbol{\mu} = \mathbf{0}$) and with covariance matrix ($\boldsymbol{\Sigma}$) the identity (2×2).

$$f(\mathbf{x}) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}{(2\pi)\sqrt{|\boldsymbol{\Sigma}|}} \tag{64}$$

The test involving the calculation of the gradient and the Hessian of gamma analytically converged quadratically to the solution, as shown in Figure 7.

On the other hand, the precision of the Hessian and the gradient when solving the ISDE with an MCMC at every iteration reaches the third decimal when used $N = 400$ and $n_{MC} = 1000$.

As seen in Figure 8, the error at the beginning decreases quadratically and then it stops decreasing when reaches the third decimal precision.

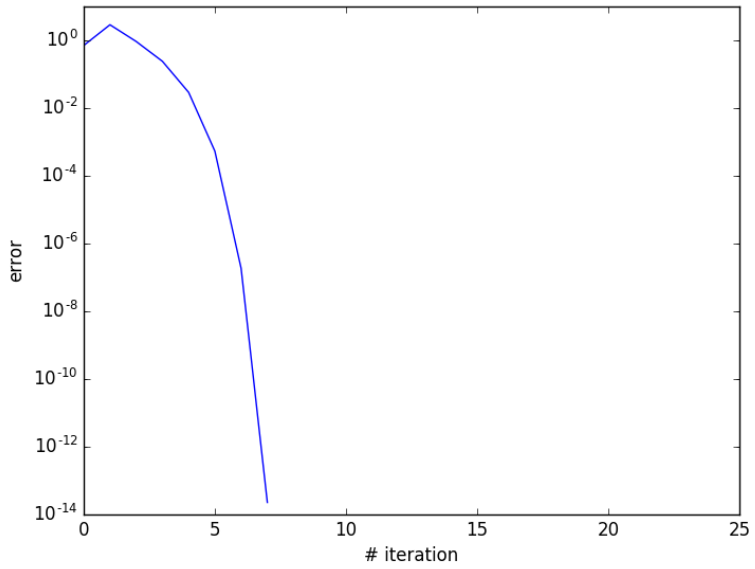

Figure 7: Convergence of the algorithm knowing the analytical solution of every iteration.

For the test where the gradient and the Hessian are calculated analytically some noise has been introduced in the analytical calculations of the gradient and the Hessian of gamma. In Figure 9 you can see first the graph of errors with added noise with the form:

- $gradient \times (1.0 + 1.0e{-}1 \times uniform[-1, 1])$ and $H \times (1.0 + 1.0e{-}1 \times uniform[-1, 1])$,

and right below it can be seen a noise of the form:

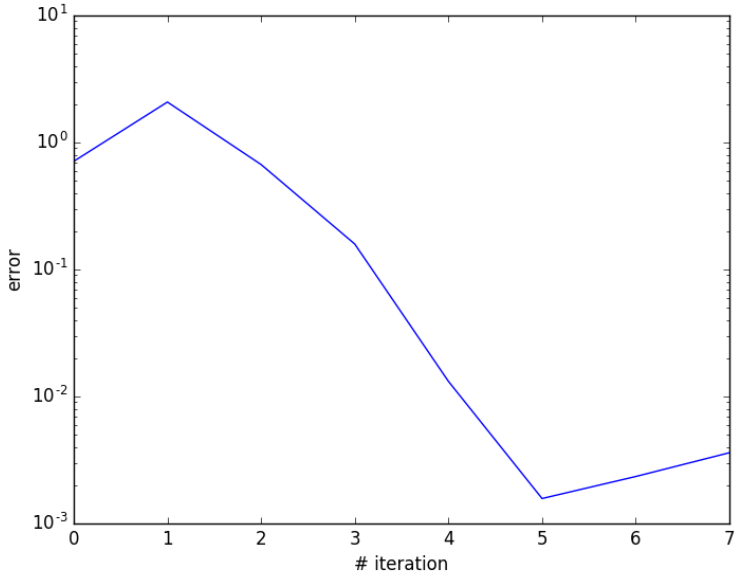

Figure 8: Convergence of the algorithm solving the ISDE with the MCMC at every iteration.

- $H + (1e-5 \times uniform[-1.e-6, 1e-6])$ and,
  $gradient + (1e-5 \times uniform[-1.e-6, 1e-6])$.

From these graphs it can be inferred that a relative error in the i-th decimal does not affect the final convergence, but the velocity of the convergence, while an absolute error of in the i-th decimal prevents the algorithm from reaching the solution in that same precision.

Figure 8 is closer to the case where there's absolute noise, in this way, the maximum accuracy reachable is the one given by the number of new realizations M. And to increase the precision 1 order of magnitude, M has to be increased a hundred times as the convergence is quadratic.

5.2.1 Convergence of the MCMC generator of the ISDE

As mentioned before, the method used to solve the discretization of the Ito Stochastic Differential Equation, expressed in (53) and the following equation, is the Störmer-Verlet, exposed in Appendix A. This method allows reaching stability in several orders of magnitude fewer than other algorithms such as the Metropolis-Hastings algorithm, as seen in the book of Soize [6].

The convergence of the Störmer-Verlet method in the previous case it is achieved very quickly. To see graphical results of it, in Figure 10, both the first and the second moment of the first component of $\mathbf{H}_\lambda$ are computed with new N realizations of $\mathbf{H}_\lambda$ computed with the Störmer-Verlet for the discretized ISDE: $\{z_i \in \mathbb{R}^2, i = 1, \ldots, N\}$.

For this it is necessary calculating for every N the following equations:

$$m(K) = \frac{1}{K} \sum_{i=1}^{K} z_{i1} \tag{65}$$

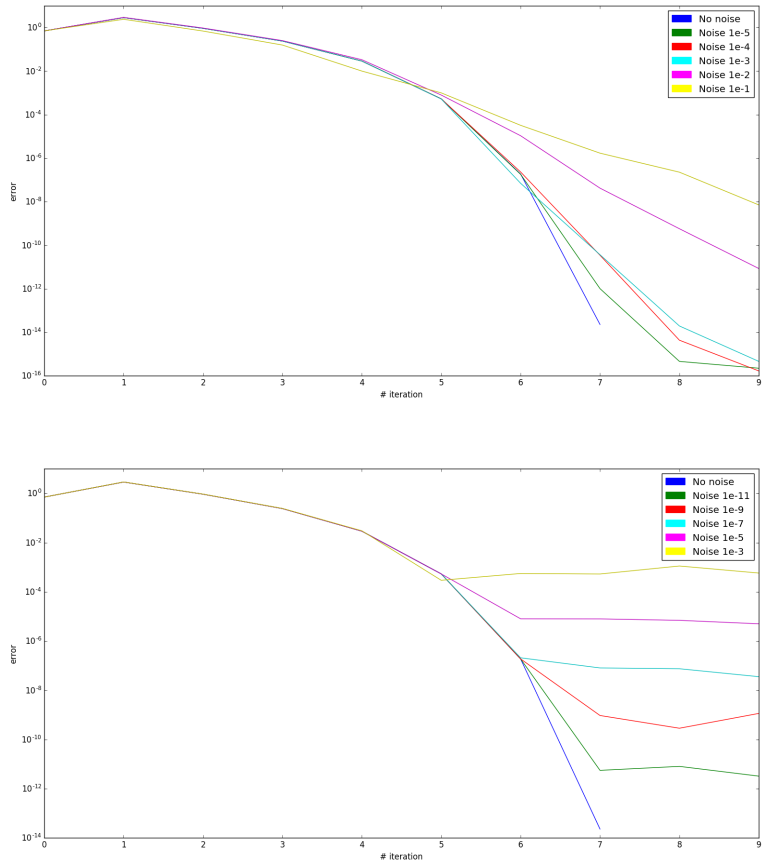

Figure 9: Comparison between relative and absolute noise.

$$m_2(K) = \frac{1}{K}\sum_{i=1}^{K} z_i^2 1 \tag{66}$$

In Figure 10 it is shown the graphs of Equations $K \to m(N)$ in the top left and $K \to m_2(N)$ in the top right, and right below, their autocovariances. The convergence is achieved with $K = 1500$ realizations.

The parameters used are the ones used for solving the empirical test mentioned in the section before, $\Delta t = 0.015s$, $f_0 = 1.5$ and without the diffusion maps.

5.2.2 Diffusion maps

The introduction of the diffusion maps basis, presented in section 4.8, has several effects on the convergence of the algorithm.

First of all, it has a direct effect on the calculation of the empirical gradient and Hessian of gamma, in the case where the constraints include some of the moments of some components. For example the first and second moment of a few of its components.

The diffusion maps basis concentrates the new realizations in a subset $\mathcal{S}_\nu$ of $\mathbb{R}^N$, which implies that the variance of the new realizations is different if the subset $\mathcal{S}_\nu$ is not well identified. It is assumed that in this subset $\mathcal{S}_\nu$ is where the probability density functions $p_{\mathbf{H}\lambda}$ are concentrated.

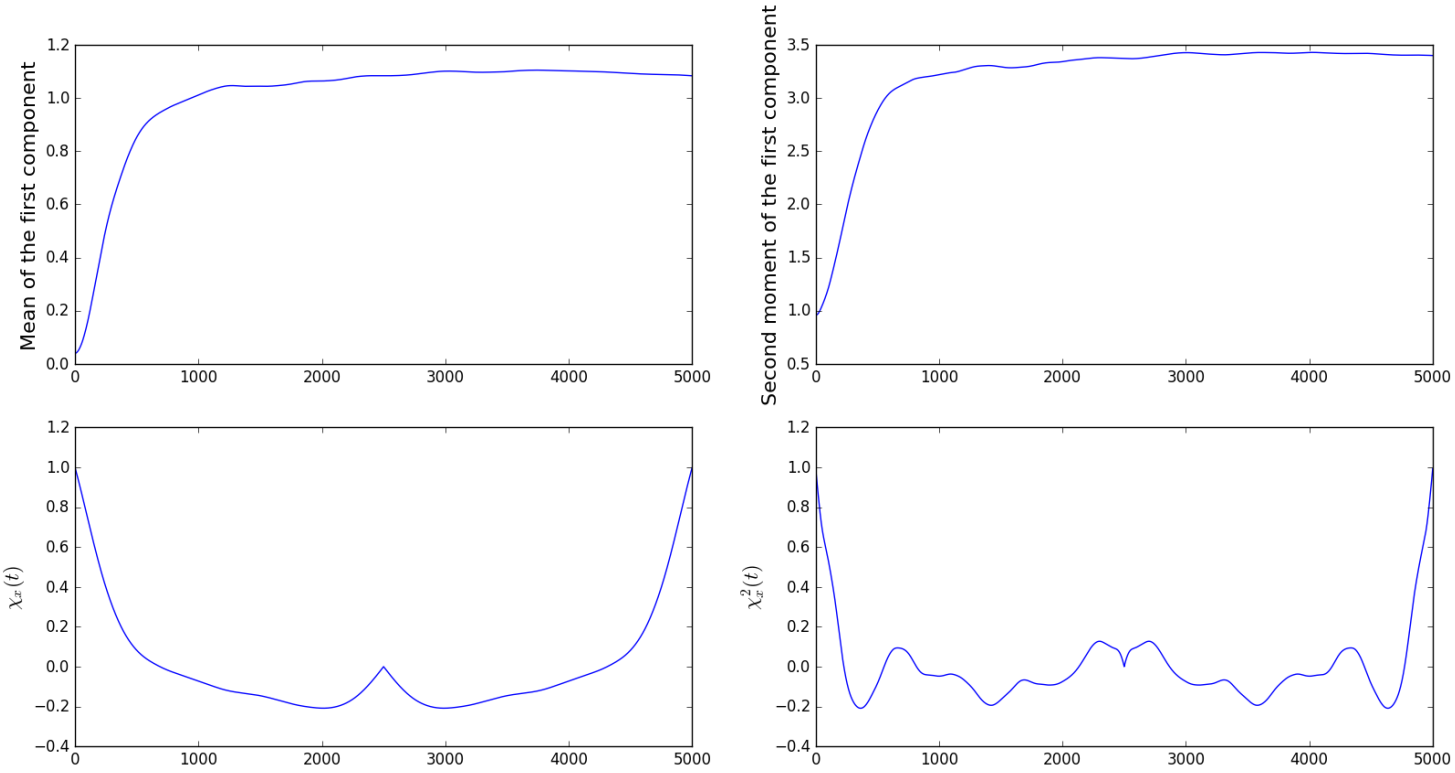

Figure 10: Graphs of Equations (65) top left and (66) top right and their autocovariances. The x-axis denotes the number of steps.

This is directly reflected in the calculation of the gradient and the Hessian matrix of gamma, defined in Equations (29) and (31). If the subset $\mathcal{S}_\nu$ is not well identified, it can mean that it concentrates the new realizations in a more concentrated subset. Having less variance due to the reduction of the diffusion maps basis implies different values of both the gradient and the Hessian.

Thus, the next step in the Newton method, Equation (32), is totally different from the one it would be if the diffusion maps basis reduction had not been applied.

To well identify the subset $\mathcal{S}_\nu$, different criteria is proposed in different works of Soize and Ghanem to identify the two parameters that determine the diffusion maps basis, m and ε.

The diffusion maps allows to achieve better accuracy with the same number of time steps for the ISDE. The manifold reduction always done by the diffusion maps basis reduces in the 'data' space (of dimension N) and not the ambient space (of dimension n). The PCA is meant to reduce the ambient space $\mathbb{R}^n$ into $\mathbb{R}^\nu$.

$[\mathbf{Z}_{\lambda^i}]$, definded in Equation (45), are the *Galerkin* projection parameters. The *Galerkin* method a basis for the unknown is defined. Then, the residual of the governing equation is such that is orthogonal to that same basis. This latter requirement then generates the equations for the parameters.

If the dimension ν is small, there may appear some issues when trying to characterize the subset $\mathcal{S}_\nu$.

In a similar example to the previous Gaussian example, however, in this case the constraints imposed are composed by the first moment for the two components ($\nu = 2$).

The plot of the sorted eigenvalues of matrix $[\mathbb{P}]$ in $\nu = 2$ descends to zero very quickly:

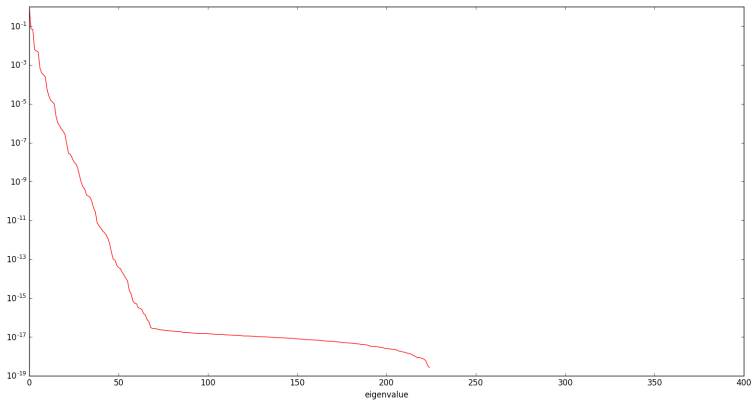

Figure 11: Plot of the sorted eigenvalues of matrix $[\mathbb{P}]$ in $\nu = 2$.

In bigger dimensions of ν the spectrum of the diffusion maps basis seems to have not a such strongly decreasing tendency and at some point it stays mostly flat. A good point to set the value of m.

In this case, where the diffusion maps does not well identify the subset $\mathcal{S}_\nu$, the algorithm diverges, if applied with the diffusion maps and the procedure to choose ε and m are applied ($\varepsilon = 4$, $m = 6$).

In Figure 12 it is seen the histogram after the first Newton iteration. In this case the diffusion maps is not able to identify $\mathcal{S}_\nu$ and concentrates too much the distribution.

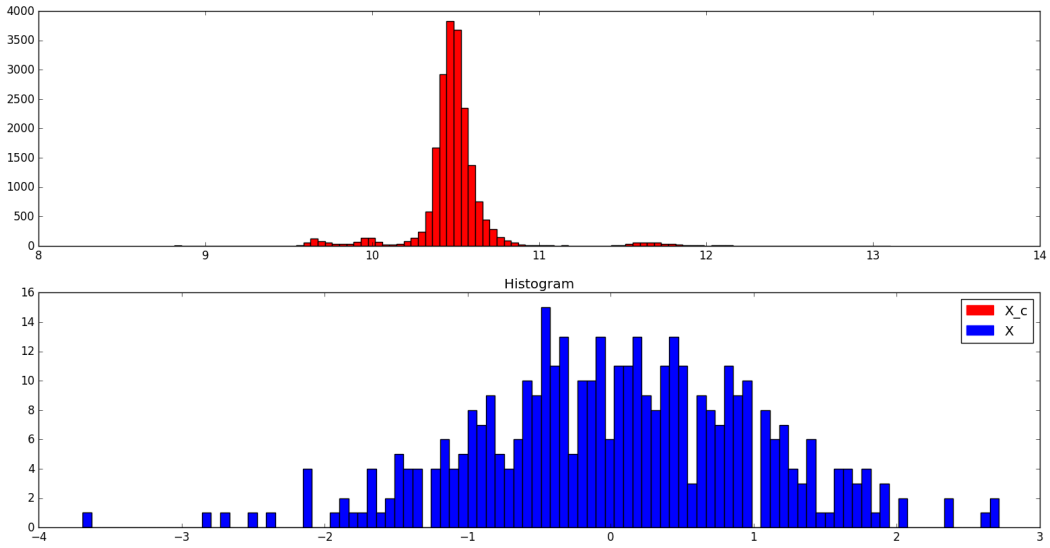

Figure 12: Histogram of the first of the two components after the first iteration of problem in $\mathbb{R}^2$ without diffusion maps.

If the algorithm is applied without diffusion maps, the final histogram is plotted as shown in Figure 13. The concentration of the new realizations ($\mathbf{X}^c$) is smaller than in the case with the diffusion maps reduction.

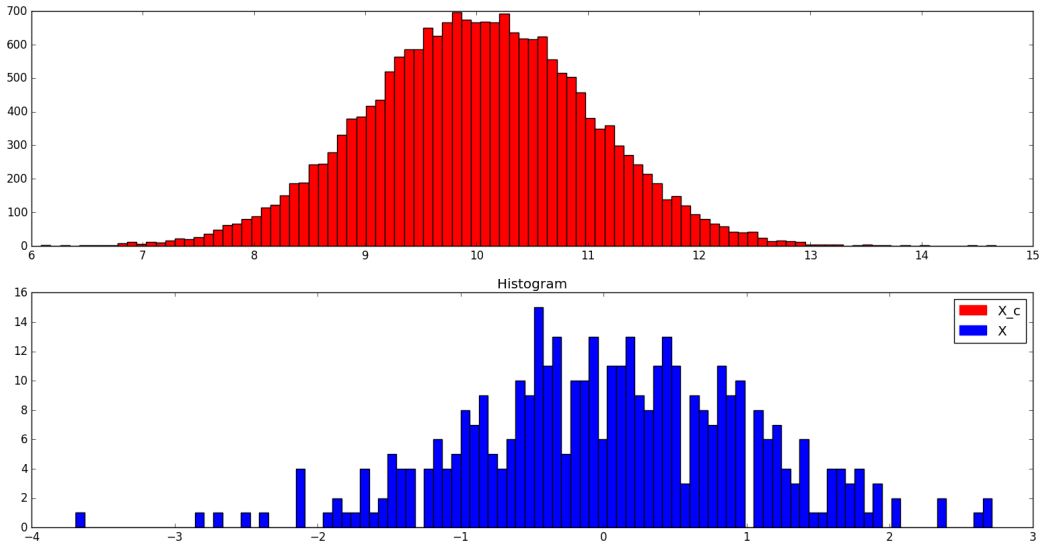

Figure 13: Final histogram of the first component of problem in $\mathbb{R}^2$ without diffusion maps.

The diffusion maps has a big computational importance due to the fact that it may reduce 10 or 100 times the number of steps taken in the MCMC to achieve a better convergence of the MCMC itself and consequently of the Newton method.

The analogy between the relation of m and the number of steps needed in the MCMC, and between N and the bandwidth for the KDE is that the smaller the dimension ν, the fewer N number of points are needed to have a smooth representation of the probability distribution. And something similar happens with m and the number of steps in the MCMC needed to achieve convergence. The smaller is m, the faster the convergence is achieved. But on the other hand, the smaller is m, the fewer information about the data set it is worked with, and such a small m can bring a huge loss of information.

The diffusion maps reduction works much better in higher dimensions, some examples at higher dimensions will be presented in Section 6.

6 Applications

Two applications have been tested in order to get more realizations and enable to do the computations they require. Also an abstract example has been performed to see the behavior of the algorithm.

The first application is an example of the algorithm without constraints. It is basically running the Newton method just in the very first iteration.

The example is a designed problem with some random distributions in $\mathbb{R}^{20}$ and some constraints that the original data points should be consistent with.

The second application also involves the integrity of the presented algorithm, in which initial data and initial constraints are given.

6.1 Application 1

This first application is a practical case in which some integrals have to be calculated using Monte Carlo methods. These methods require a large amount of realizations to have an accurate result of the integrals.

The issue is that calculating new realizations requires an unaffordable amount of time and a large computational cost.

The integrals to be computed are part of a variance based sensitivity analysis, presented in the work of Andrea Saltelli et al. [7], of a model of Lithium Ion Batteries.

From a finite element method model, that models a Li-Ion battery, we seek to understand the impact of the input model parameters.

For every input, the output of this model has to be computed, requiring a large amount of time. In this way the PLoM without constraints comes into a role, where it will generate more pairs of the form (input, output). Having much more pairs the Monte Carlo integrals will be much more precise than with few representations of this inputs.

This variance based sensitivity analysis, presented in the work of Andrea S. [7], aims to study how the variation of the inputs affect the outputs in this finite element model and seek to understand the impact of the input model parameters.

Say we have an output function $\mathrm{y} = f(\mathrm{x})$ with d input parameters $\mathbf{x} = (x_1, x_2, \ldots, x_d)$. f: $\mathbb{R}^d \to \mathbb{R}$. It is defined:

1. Local sensitivity:
$$S_i = \frac{\delta f}{\delta x_i} \tag{67}$$

2. Global sensitivity (Variance based):
$$S_i = \frac{V_{x_i}\left(E_{\mathrm{x}_{-i}}\left(y|x_i\right)\right)}{V(y)} \tag{68}$$
$$S_i^T = \frac{E_{\mathrm{x}_{-1}}\left(V_{x_i}\left(y|\mathbf{x}_{-i}\right)\right)}{V(y))} = 1 - \frac{V_{\mathrm{x}_{-1}}\left(E_{x_i}\left(y|\mathbf{x}_{-i}\right)\right)}{V(y)} \tag{69}$$
where $\mathbf{x}_{-i}$ denotes the vector of all factors but x_i

The sensitivity test carried out requires the generation of samples inputs and the calculation of some integrals in the following way:

- Generate an $N \times 2d$ sample matrix, where N is the number of Monte Carlo trials and d is the input parameter range.
- Construct 3 matrices from this: the first set of d columns for $\mathbf{A}$, the second set of d columns for $\mathbf{B}$, and $\mathbf{A}_B^i$. Where $\mathbf{A}_B^i$ is the matrix $\mathbf{A}$ but having replaced the i-th column of $\mathbf{A}$ with the i-th column of $\mathbf{B}$.
- This will yield the FEM model outputs of $f(\mathbf{A})$, $f(\mathbf{B})$. and $f(\mathbf{A}_B^i)$:

$$f_0 = \frac{1}{N} \sum_{k=1}^{N} f(\mathbf{A})_k \tag{70}$$

$$V(y) = \frac{1}{N} \sum_{k=1}^{N} f(\mathbf{A})_k^2 - f_0^2 \tag{71}$$

$$V_{x_i} \left(E_{\mathbf{x}_{-i}} \left(y | x_i \right) \right) = \frac{1}{N} \sum_{k=1}^{N} f(\mathbf{B})_k \left(f \left(\mathbf{A}_B^i \right)_k - f(\mathbf{A})_k \right) \tag{72}$$

$$E_{\mathbf{x}_{-i}} \left(V_{x_i} \left(y | \mathbf{x}_{-i} \right) \right) = \frac{1}{2N} \sum_{k=1}^{N} \left(f(\mathbf{A})_k - f \left(\mathbf{A}_B^i \right)_k \right)^2 \tag{73}$$

Where $f(\mathbf{A})_k$ denotes the output of f when evaluated with the k-th row of matrix $\mathbf{A}$: $f(\mathbf{A}_k)$.

As a result, the points we want to reproduce will have an structure as follows,

Function f has a high computational cost, and just N realizations have been computed with the FEM model. The algorithm without constraints allows to generate M new realizations of $\mathbf{X}$, that follows the unknown distribution of the columns of the structure in Table 2.

In this application $d = 4$ and $N = 875$. In Figure 15 the rows corresponding to $f(\mathbf{A})$ and to $f(\mathbf{B})$ are plotted.

The inputs to the function $f(x_1, \ldots, x_d)$ are material constants in the battery model. x_1 and x_2 are the reaction rate constants for the anode and cathode, and x_3 and x_4 are the diffusion coefficients for the cathode and anode, respectively. They have a similar importance as the Young's modulus / Poisson ratio in solid mechanics, as they are pretty important in the battery's performance.

The reaction rate will relate how fast the lithium ions transfer from the electrolyte to the electrode or vice versa during charge/discharge. The diffusion coefficient is directly proportional to the flux of lithium ions within the electrode itself.

Figure 14 is an sketch of the finite element mesh of the battery.

The output is the "battery capacity" of the full cell, as seen in Equation (74). When a battery is being discharged, there is an electric current that is outputted which can then be used to power devices like phones, laptops, etc. For a certain time interval it is being extracted the electric flux value from the finite element model (FEM), and then doing a Riemann sum over time intervals with the flux value to determine the capacity.

So if the electric flux is called "iflux" and dt is called the length of the time interval, the Riemann sum is just extended over how long it is running the simulation and then multiplying iflux$\times dt$ and summing over.

$\mathbf{A}^T$ (d rows $\times$ N columns)			
$\mathbf{B}^T$ (d rows $\times$ N columns)			
$f(\mathbf{A})_1$	$\dots f(\mathbf{A})_k \dots$	$f(\mathbf{A})_N$	
$f(\mathbf{B})_1$	$\dots f(\mathbf{B})_k \dots$	$f(\mathbf{B})_N$	
$f\left(\mathbf{A}_B^1\right)_1$	$\dots f\left(\mathbf{A}_B^1\right)_k \dots$	$f\left(\mathbf{A}_B^1\right)_N$	
$\vdots$	$\vdots$	$\vdots$	
$f\left(\mathbf{A}_B^i\right)_1$	$\dots f\left(\mathbf{A}_B^i\right)_k \dots$	$f\left(\mathbf{A}_B^i\right)_N$	$(1 \leq i \leq d)$ $(1 \leq k \leq N)$
$\vdots$	$\vdots$	$\vdots$	
$f\left(\mathbf{A}_B^d\right)_1$	$\dots f\left(\mathbf{A}_B^d\right)_k \dots$	$f\left(\mathbf{A}_B^d\right)_N$	

Table 2: Structure of the N initial points in $\mathbb{R}^{3d+2}$

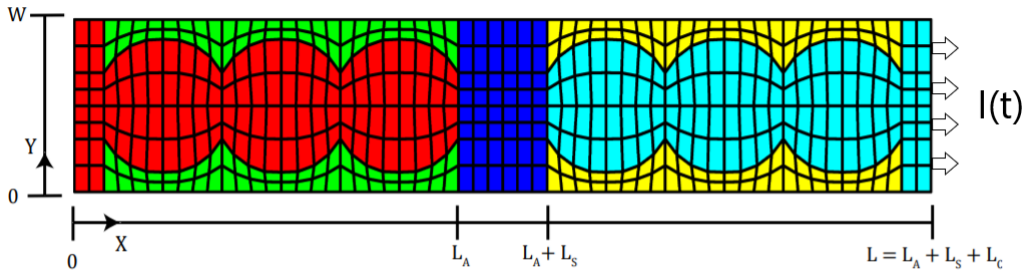

Figure 14: Sketch of the battery with anode (red), cathode (light blue), electrolyte separator (dark blue), surrounding electrolyte (green and yellow).

$$f(x_1, \dots, x_d) = Q = \int_{t_1}^{t_2} I(t, x_1, \dots, x_d) dt \tag{74}$$

After compting the PCA with a tolerance of $1e-3$, the new dimension is $\nu = 10$. The parameters that define the diffusion maps basis are:

- $\varepsilon = 30$.
- $m = 12$
- Kernel function: the same function defined in section 4.8.

And finally, to solve the reduced ISDE, with the Störmer-Verlet algorithm, the parameters considered are:

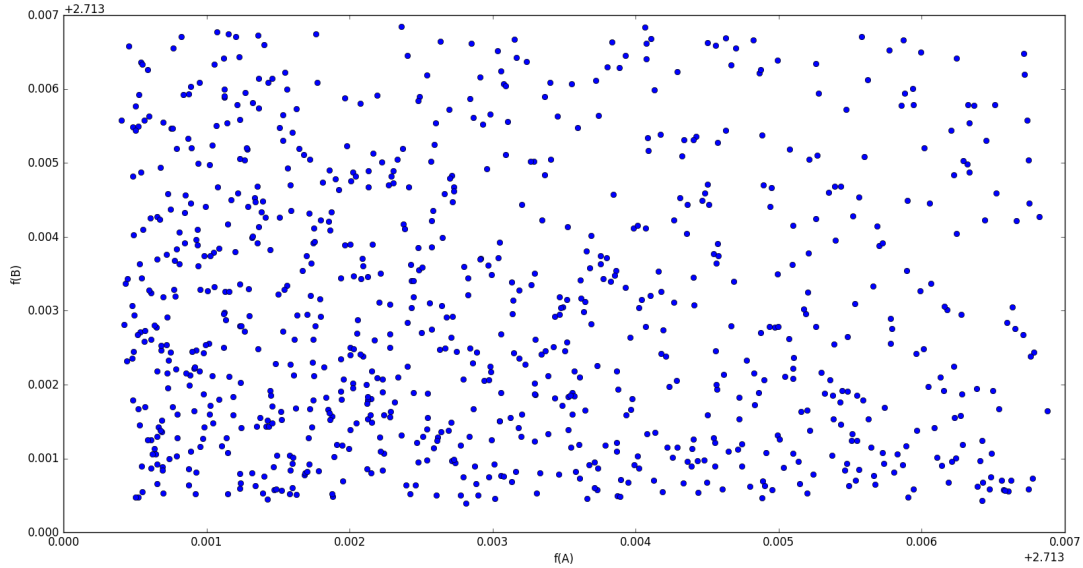

Figure 15: Plot of the components $f(\mathbf{A})$ and $f(\mathbf{B})$ of the initial data set.

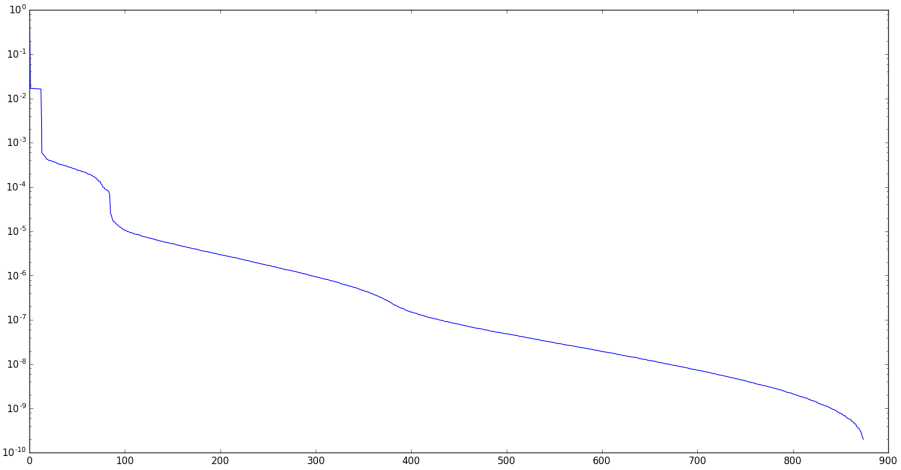

Figure 16: Sorted plot in log-scale of the eigenvalues of matrix $[\mathbb{P}]$.

M_0	10
Δt	0.1627
f_0	1.5
n_{MC}	200

Table 3: Parameters of the MCMC.

M_0 denotes the number of steps between picking two points of the Markov Chain.

As a result of parameters defined previously, the new number of realizations $M = N \times n_{MC} = 875 \times 200 = 175000$. This number is more than enough to calculate the integrals using Monte Carlo methods, as depicted in Equations (70) to (73).

Next, the following histograms of several components show the coherence of the new realizations when compared with the original data set.

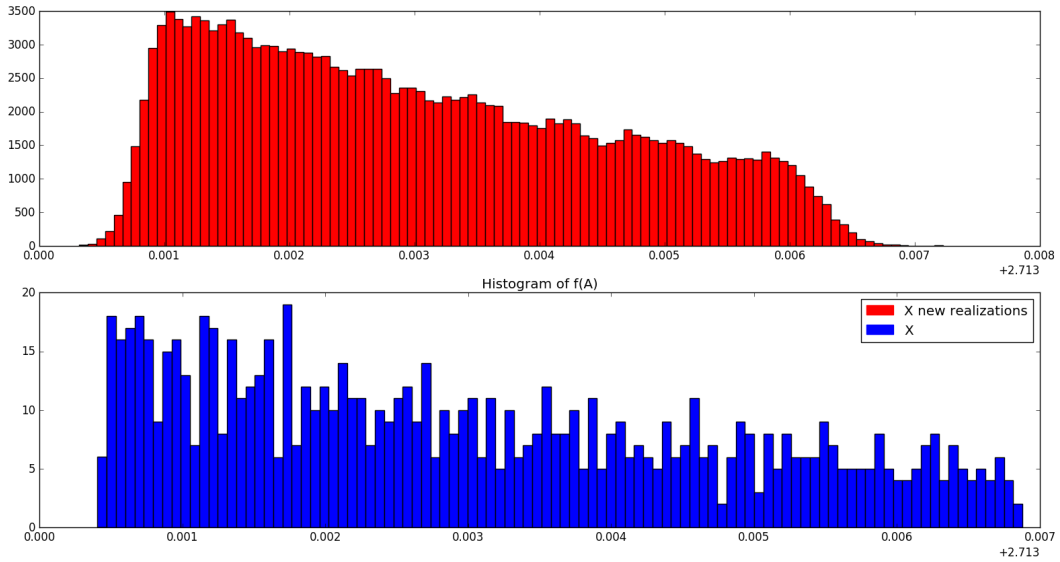

Figure 17: Histogram of the component f(A), both the initial data set and the new realizations.

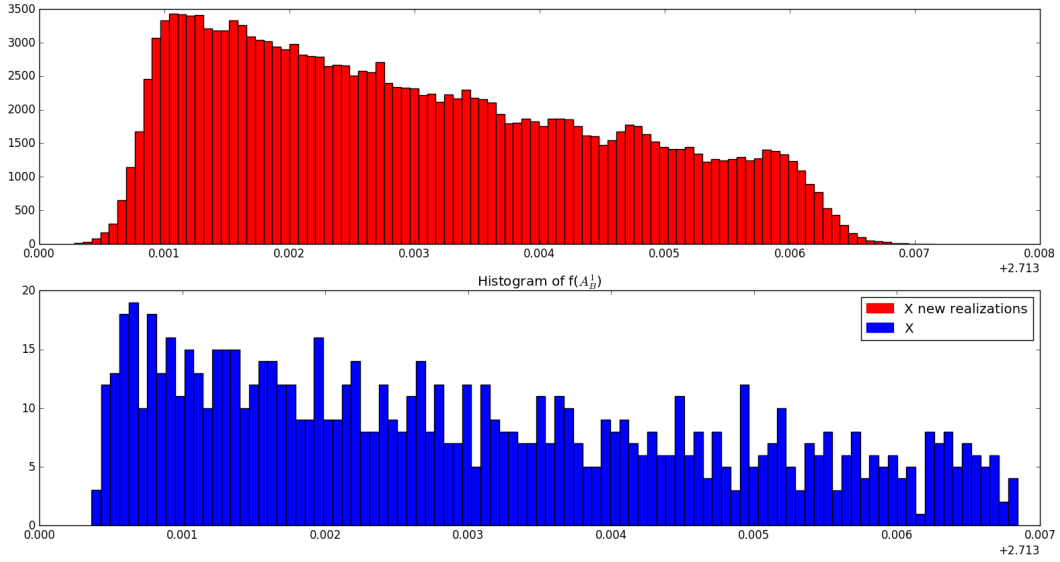

Figure 18: Histogram of the component f(A_B^1), both the initial data set and the new realizations.

The first and second moment of the initial data set and the new realizations are the same for all the 14 components of the points.

At first, with just 875 points the results of the equations (70) to (73) led to illogical results. With the new 175,000 realizations, the results are understandable. The 875 points where not a representative realization of the space to be integrated.

The limitations of the Finite Elements Method model are evident, to generate 875 samples more than two days have been necessary to finish the calculations. On the other hand, the 175,000 realizations have peen computed in less than 10 minutes, and obtaining better results.

6.2 Example of the PLoM with constraints

In this section a theoretical example for the PLoM with constraints will be presented. It will be shown the results and the convergence of the algorithm, specially the convergence of the Newton method.

We are considering the following problem:

- $\mathbb{X} \in \mathbb{R}^{20}$
- The initial data set is composed by 300 initial points, in which the first 5 components follow specific random distributions expressed below. The components from 6 to 20 are variations of the 4th component:

```
N = 300
x = np.zeros((20,N))
np.random.seed(6)
r = np.random.weibull(3, N)+10*np.ones((1,N))
angle = np.random.uniform(0, 2*pi, N)
x[0] = r*np.sin(angle)+ 10*np.ones((1,N))
x[1] = r*np.cos(angle)
x[2] = r*r*np.cos(angle*angle)
x[3] = np.random.rayleigh(0.5, N)
x[4] = -2*x[3] + np.random.uniform(-0.1, 0.1, N)
for i in range(5,20):
      x[i] = ((−1)^2)*i*x[3]+ np.random.uniform(-0.1, 0.1, N)
```

- Then, variable $\mathbb{X}$ is scaled, as expressed in equation (5) of the paper of Soize and Ghanem [1]. This step is important because it gives to every component the same importance when computing the PCA:

$$\mathbb{X} = [\alpha_x]\,\mathbf{X} + x^{\min}, \quad \mathbf{X} = [\alpha_x]^{-1}\left(\mathbb{X} - x^{\min}\right) \tag{75}$$

- The data for the problem is: $\mathbf{X}$, and the following constraints for $\mathbb{X}$:

$$\int_{\mathrm{R}^n} \begin{pmatrix} x_1 \\ \\ x_{11} \end{pmatrix} \hat{p}_(\mathbb{X})(x) d\mathbf{x} = \begin{pmatrix} 10 \\ \\ 6 \end{pmatrix} \tag{76}$$

- The tolerance for the PCA is 1e-5. After computing the PCA and the diffusion maps, we get using the steps mentioned in the previous sections: $\nu = 8$ and $m = 10$; i.e. $20 \mapsto 8$ and $300 \mapsto 10$

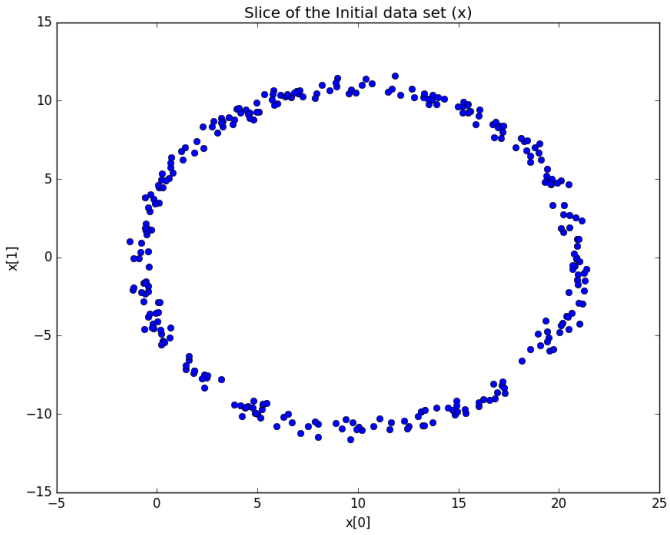

Figure 19: Plot of $x[0]$ and $x[1]$.

The spectrum of matrix $\mathbb{P}$, plotted in Figure 20, seems much more reasonable than in the example of section 5.2.2. Choosing $m = 10$ is consistent with the fact of keeping enough information but not discarding too much of it, since the eigenvalues $\Lambda_j, j > 10$ are considerably smaller than the first eigenvalues of the spectrum.

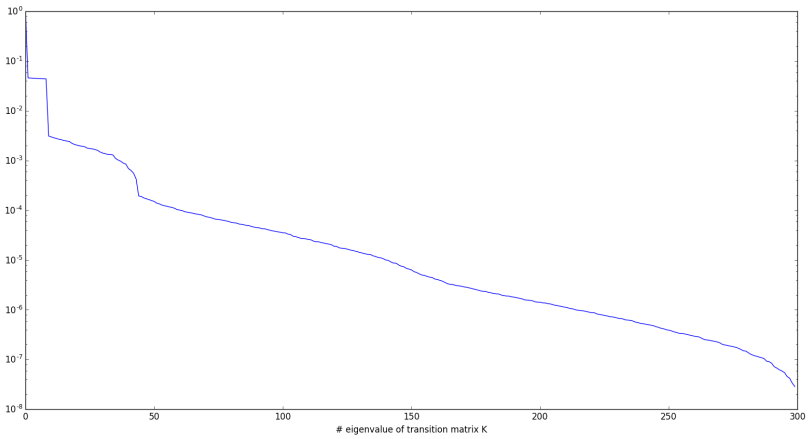

Figure 20: Plot in log-scale of $[\mathbb{P}]$ spectrum.

And finally, to solve the reduced ISDE, with the Störmer-Verlet algorithm, the parameters considered are shown in Table 4. M_0 denotes the number of steps between picking two points of the Markov Chain. l_0 denotes the number of steps omitted at the beginning of the Markov Chain.

M_0	10
l_0	10
Δt	0.156
f_0	1.5
n_{MC}	50

Table 4: Parameters of the MCMC of the problem in $\mathbb{R}^{20}$.

The convergence of the Störmer-Verlet is achieved at the 10th step, as seen in Figure 21.

$$m_1(K) = \frac{1}{K}\sum_{i=1}^{K} x_{i1} \tag{77}$$

$$m_{11}(K) = \frac{1}{K}\sum_{i=1}^{K} x_{i11} \tag{78}$$

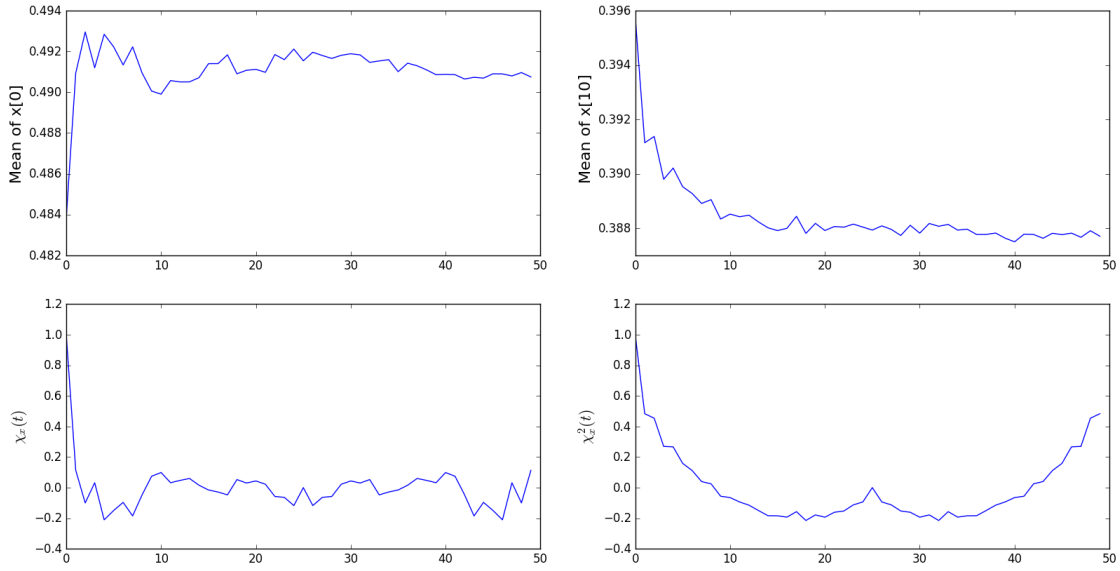

Figure 21: Graphs of Equations (77) top left and (78) top right and their autocovariances. The x-axis denotes the number of steps.

The convergence of the newton method seems to be exponential. To get more accuracy much more iteration would be needed. The initial condition of the Newton method is already considerably small, and it is just reduced to the order of 1e-3. In the error plot, in Figure , the error describes an exponential, and in the log-scale plot, it has a straight line tendency.

Although the proposed constraints might seem easy, for this problem they may be more complicated than expected, specially when it comes to calculate the new realizations and their associated gradient and hessian.

This might be one of the determining aspects for the Newton method convergence.

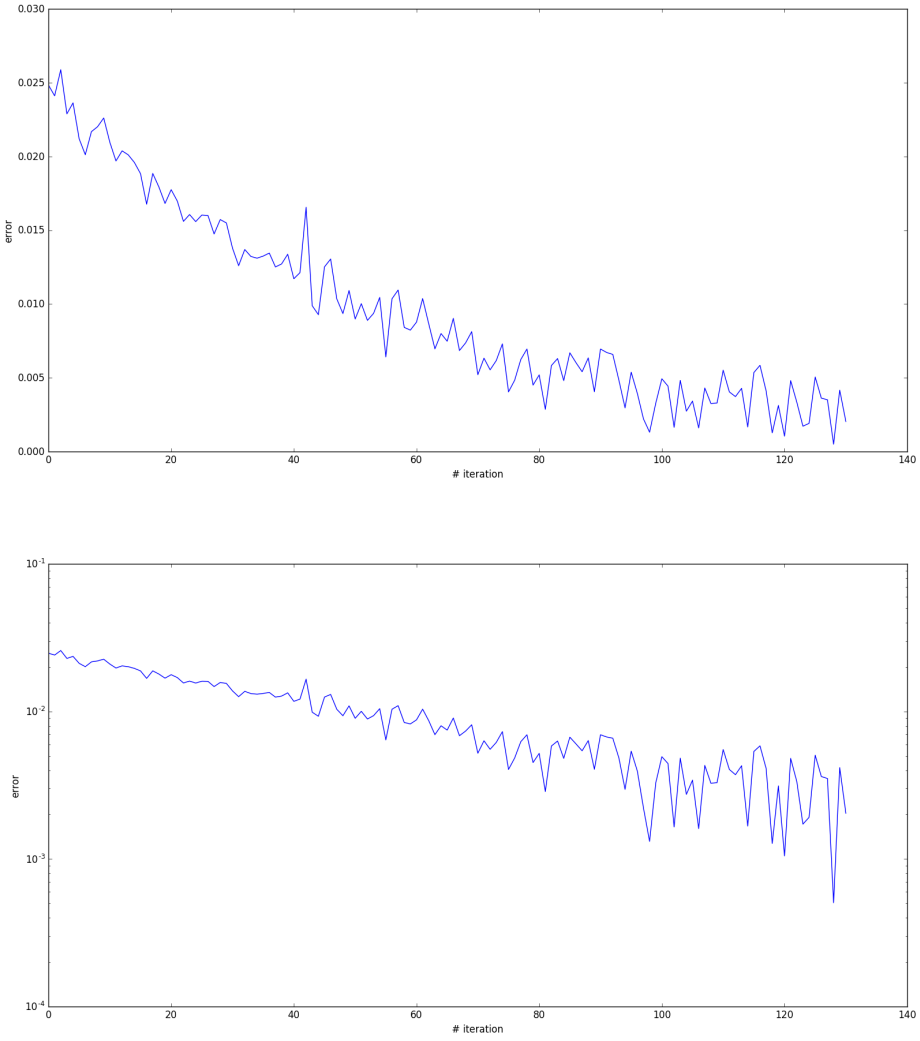

Figure 22: Plot of the error function in natural and logarithmic scale.

In Figure 23 it can be seen the distribution of the initial data set and the new realizations, which are consistent with the constraints expressed in Equation (76).

The diffusion maps reduction seems to keep the dispersion probability distribution of the initial data set. And the reduction from $300 \mapsto 10$ keeps enough information for the convergence of the algorithm.

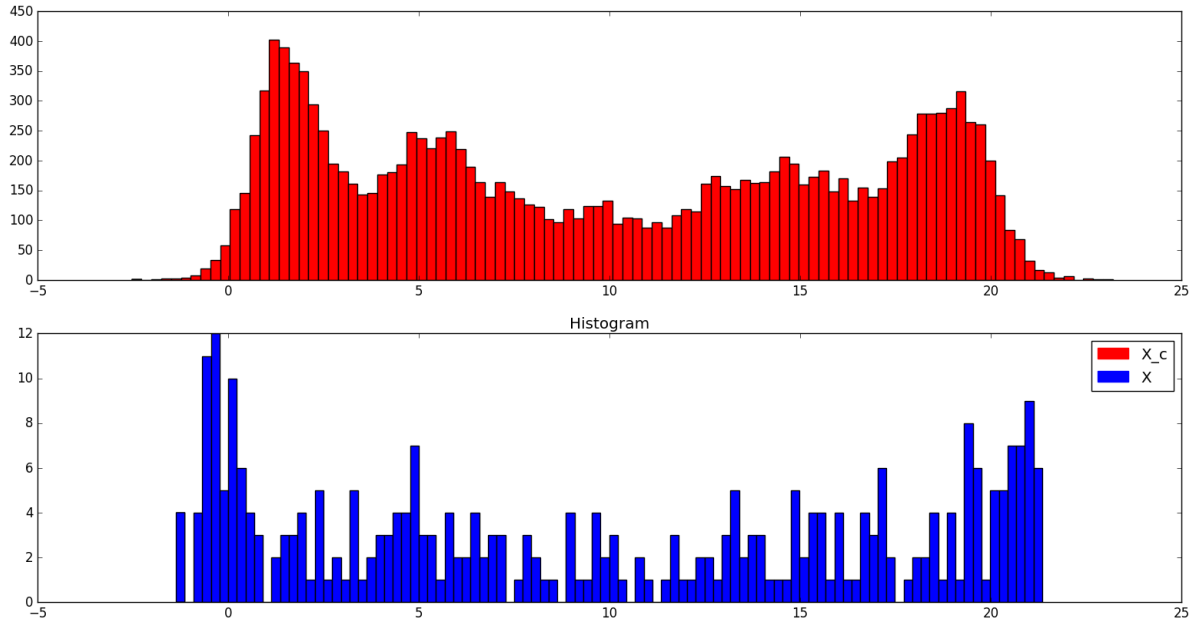

Figure 23: Histogram of the first component of $\mathbb{X}$ and $\mathbb{X}^c$, both of the initial data set (X) and the new realizations consistent with the constrains (X_c).

6.3 Application 2

The second application consists of the same problem as in Application 1, however, in this second application some constraints are given.

In this case the dimensions shown in Table 2 are slightly different. In this case $d = 9$ and $N = 1120$. Where the input parameters of f: $\mathbb{R}^d \to \mathbb{R}$ mean:

1. The reaction rate constant of the anode.
2. The reaction rate constant of the cathode.
3. The diffusion coefficient of the cathode.
4. The diffusion coefficient of the anode.
5. The diffusion coefficient of the electrolyte.
6. The electrical conductivity of the anode.
7. The electrical conductivity of the electrolyte.
8. The electrical conductivity of the cathode.
9. The temperature at which the simulation is running.

Also, to the matrix shown in Table 2, $f\left(\mathbf{B}_{AB}^{i}\right)_k$, the output of the reciprocal matrix of the input matrix of $f\left(\mathbf{A}_{B}^{i}\right)_k$. In this way, the initial data set has the shape shown in Table 5.

$\mathbf{A}^T$ (d rows $\times$ N columns)			
$\mathbf{B}^T$ (d rows $\times$ N columns)			
$f(\mathbf{A})_1$	$\dots f(\mathbf{A})_k \dots$	$f(\mathbf{A})_N$	
$f(\mathbf{B})_1$	$\dots f(\mathbf{B})_k \dots$	$f(\mathbf{B})_N$	
$f\left(\mathbf{A}_B^1\right)_1$	$\dots f\left(\mathbf{A}_B^1\right)_k \dots$	$f\left(\mathbf{A}_B^1\right)_N$	
$\vdots$	$\vdots$	$\vdots$	
$f\left(\mathbf{A}_B^i\right)_1$	$\dots f\left(\mathbf{A}_B^i\right)_k \dots$	$f\left(\mathbf{A}_B^i\right)_N$	$(1 \leq i \leq d)$
$\vdots$	$\vdots$	$\vdots$	$(1 \leq k \leq N)$
$f\left(\mathbf{A}_B^d\right)_1$	$\dots f\left(\mathbf{A}_B^d\right)_k \dots$	$f\left(\mathbf{A}_B^d\right)_N$	
$f\left(\mathbf{B}_A^1\right)_1$	$\dots f\left(\mathbf{B}_A^1\right)_k \dots$	$f\left(\mathbf{B}_A^1\right)_N$	
$\vdots$	$\vdots$	$\vdots$	
$f\left(\mathbf{B}_A^i\right)_1$	$\dots f\left(\mathbf{B}_A^i\right)_k \dots$	$f\left(\mathbf{B}_A^i\right)_N$	$(1 \leq i \leq d)$ $(1 \leq k \leq N)$
$\vdots$	$\vdots$	$\vdots$	
$f\left(\mathbf{B}_A^d\right)_1$	$\dots f\left(\mathbf{B}_A^d\right)_k \dots$	$f\left(\mathbf{B}_A^d\right)_N$	

Table 5: Structure of the N initial points in $\mathbb{R}^{4d+2}$

The constraints given are some means imposed in some components, that should physically follow:

$$\int_{\mathrm{R}^n} \begin{pmatrix} x_6 \\ x_8 \end{pmatrix} \hat{p}_(\mathbb{X})(x) d\mathbf{x} = \begin{pmatrix} 155 \\ 51 \end{pmatrix} \tag{79}$$

After computing the PCA with a tolerance of $1e-3$, the new dimension is $\nu = 19$. Figure 24 shows the error assumed if taken $\nu = \#eigenvalue$ in the PCA.

The parameters that define the diffusion maps basis are:

- $\varepsilon = 10$.
- $m = 21$
- Kernel function: the same function defined in section 4.8.

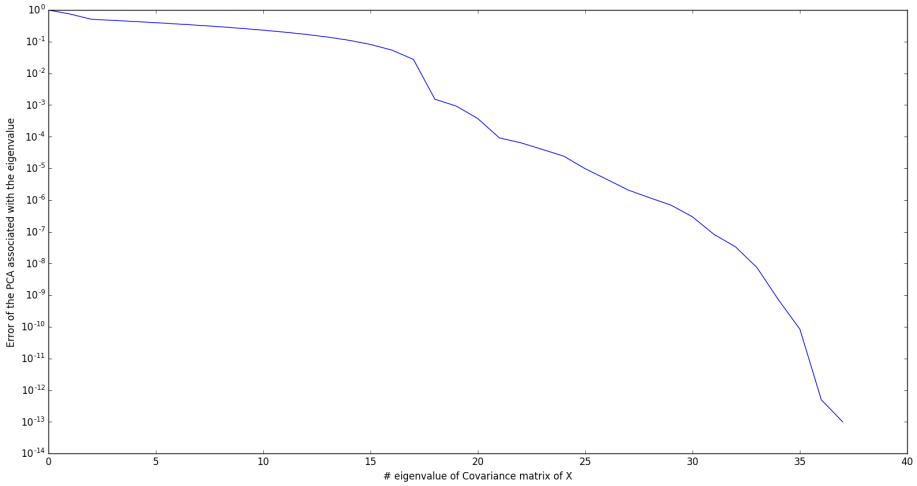

Figure 24: Graph in log-scale of the error in the PCA.

And finally, to solve the reduced ISDE, with the Störmer-Verlet algorithm, the parameters considered are:

M_0*	10
Δt	0.177
f_0	1.5
n_{MC}	200

Table 6: Parameters of the MCMC.

*M_0 denotes the number of steps between picking two points of the Markov Chain.

As a result and having the parameters defined previously, the new number of realizations $M = N \times n_{MC} = 1120 \times 200 = 224000$. This number is more than enough to calculate the integrals using Monte Carlo methods, as depicted in Equations (70) to (73).

The final convergence, the convergence of the Newton method, seems to reach a good precision in fewer steps than in the previous section. In Figure 25 it can be seen the error function in both normal and log-scale.

In the graph it cannot be seen an exponential behavior as much as in the previous example, however, the precision in the 5th decimal is achieved in fewer steps.

In Figure 26, it can be seen that the 5th component of random variable $\mathbb{X}^c$ is centerede in 155 as shown in Equation (79).

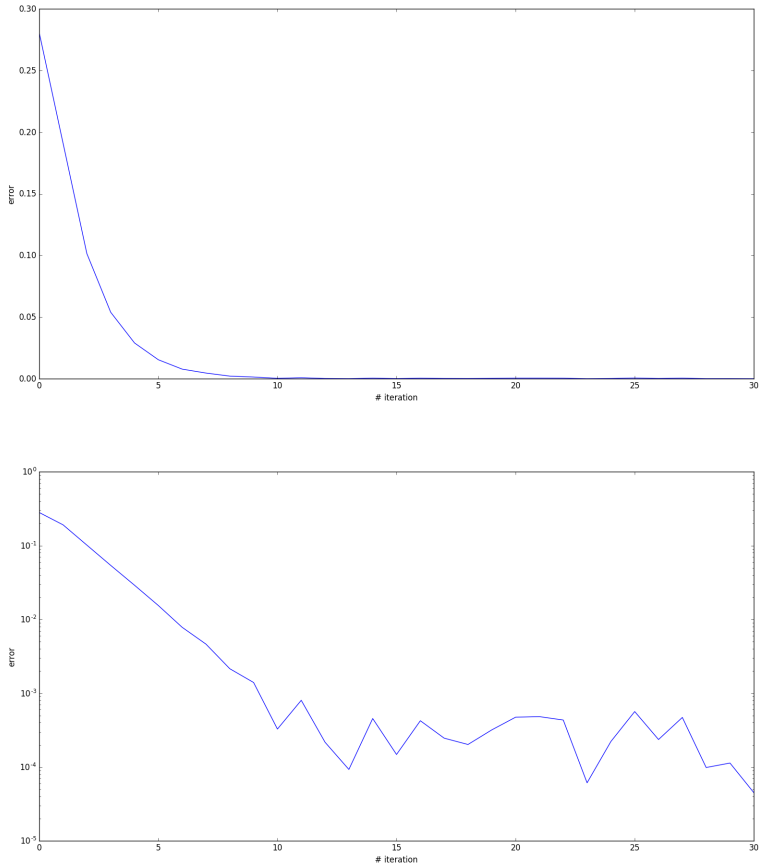

Figure 25: Graph of the error in the Newton method.

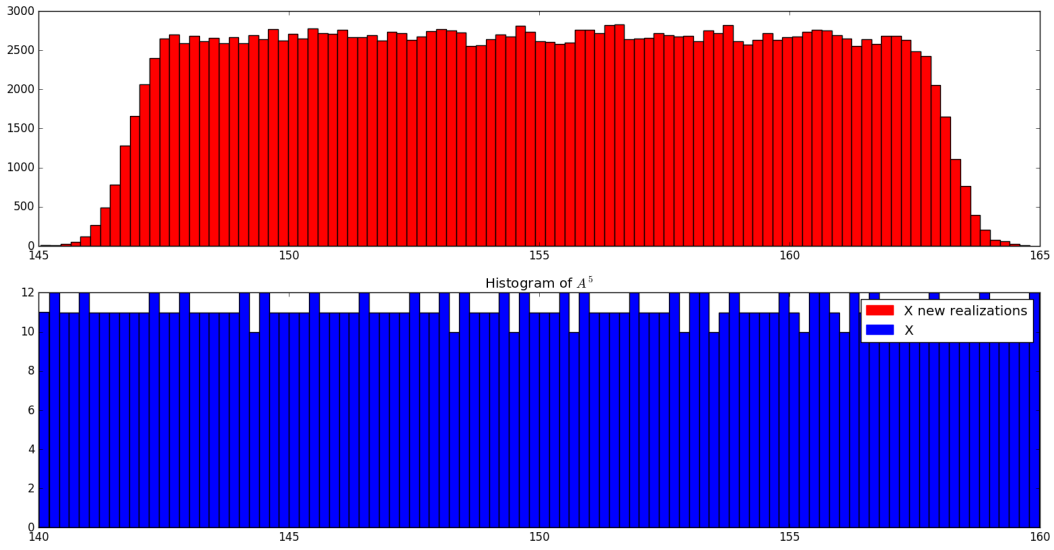

Figure 26: Histogram of $\mathbf{A}_5$, the fifth row of $\mathbf{A}$, centered in 155.

7 Results of the applications

Even though the algorithm puts together many steps, it seems to converge with relative accuracy. Although it is slow, in these cases it seems to converge exponentially. Newton method in convex problem converges quadratically, but the fact of calculating both the gradient and the hessian matrix empirically modify this convergence.

For some applications this accuracy may be enough, but other would require much more digits of accuracy.

The accuracy mainly depends on how the subset $\mathcal{S}_\nu$ is detected. All the other steps are able to achieve big precision already. In this way, the characterization of the different parameters that determine the diffusion maps basis gain special importance when it comes to accuracy.

8 Future works

The next steps would be doing a deeper research on the adjustment of the parameters to obtain better convergences in these kind of problems. There are other papers, by Soize and Ghanem, in which other ways of calculating some of the parameters are presented.

The study of the characterization of the diffusion maps basis is also a section of the algorithm to deepening more. Specially to understand its behavior in certain problems, specially in low dimensional problems.

It would also be convenient test the algorithm with an artificial intelligence algorithm. Generate more data for a machine learning model and study the accuracy of the predictions.

This algorithm can be applied to any sort of probability distributions and as mentioned before, the constraints are not just restricted to moments of components. This leads to an endless list of applications that may need more realizations to conclude some results.

9 Conclusion

In this paper several aspects of the PLoM with constraints have been analyzed. It combines many different procedures to achieve the final goal, the realization of new data realizations that follow certain restrictions. And all this steps have to come together to converge.

At practice it seems that the algorithm converges exponentially, but if the constraints aren't consistent with the initial data set, the algorithm will diverge.

The initial conditions are not limited to any particular distributions, and the constraints are not just restricted to moments.

To achieve a better accuracy in the iteration algorithm it is very important the adjustment of the parameters and the identification of subset $\mathcal{S}_\nu$.

List of Figures

List of Tables

References

[1] Soize C and Ghanem R. Physics-constrained non-gaussian probabilistic learning on manifolds., 2020.

[2] Soize C. *The Fokker-Planck Equation for Stochastic Dynamical Systems and its Explicit Steady State Solutions.* Singapore: World Scientific Publishing, 1994.

[3] Christian Soize and Roger Ghanem. Probabilistic learning on manifolds, 2020.

[4] Ofir Lindenbaum, Arie Yeredor, Moshe Salhov, and Amir Averbuch. Multiview diffusion maps, 2015.

[5] Coifman R; Lafon S; Lee A; et al. Geometric diffusions as a tool for harmonic analysis and structure definition of data: diffusion maps., 2005.

[6] C. Soize. *Uncertainty Quantification: An Accelerated Course with Advanced Applications in Computational Engineering.* New York, NY:Springer; 2017, 2017.

[7] Ivano Azzini Francesca Campolongo Marco Ratto Stefano Tarantola Andrea Saltelli, Paola Annoni. Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index., 2010.

Appendix A Störmer-Verlet

The Störmer-Verlet is the chosen discretization to solve the reduced ISDE, Equation (53) and (54). The algorithm is presented in Equation (80). The method is fully explained in the work of Soize and Ghanem [1].

The following equations describe the steps of the method:

$$\begin{aligned}
\left[\mathcal{Z}_{\ell+\frac{1}{2}}\right] &= [\mathcal{Z}_\ell] + \tfrac{\Delta t}{2}\,[\mathcal{Y}_\ell] \\
[\mathcal{Y}_{\ell+1}] &= \tfrac{1-\beta}{1+\beta}\,[\mathcal{Y}_\ell] + \tfrac{\Delta t}{1+\beta}\left[\mathcal{L}_{\ell+\frac{1}{2}}\right] + \tfrac{\sqrt{f_0}}{1+\beta}\left[\Delta \mathcal{W}^{\text{wien}}_{\ell+1}\right] \\
[\mathcal{Z}_{\ell+1}] &= \left[\mathcal{Z}_{\ell+\frac{1}{2}}\right] + \tfrac{\Delta t}{2}\,[\mathcal{Y}_{\ell+1}]
\end{aligned} \tag{80}$$

where,

$$[\mathcal{Z}_\ell] = [\mathcal{Z}_{\lambda'}(t_\ell)]\,, [\mathcal{Y}_\ell] = [\mathcal{Y}_{\lambda'}(t_\ell)]\,, \text{ and } \left[\mathcal{W}^{\text{wien}}_\ell\right] = \left[\mathcal{W}^{\text{wien}}_{\lambda'}(t_\ell)\right] \tag{81}$$

$$\left[\mathcal{L}_{\ell+\frac{1}{2}}\right] = \left[\mathcal{L}_{\lambda'}\left(\left[\mathcal{Z}_{\ell+\frac{1}{2}}\right]\right)\right] = \left[L_{\lambda'}\left(\left[\mathcal{Z}_{\ell+\frac{1}{2}}\right][g]^T\right)\right][a] \tag{82}$$

$$\left[\Delta \mathcal{W}^{\text{wien}}_{\ell+1}\right] = \left[\Delta \mathbf{W}^{\text{wien}}_{\ell+1}\right][a] \tag{83}$$

which holds:

$$E\left\{\left[\Delta \mathbf{W}^{\text{wien}}_{\ell+1}\right]_{\alpha j}\left[\Delta \mathbf{W}^{\text{wien}}_{\ell+1}\right]_{a'j'}\right\} = \Delta t\,\delta_{\alpha a'}\delta_{jj'} \tag{84}$$