An analogy between solving Partial Differential Equations with Monte-Carlo schemes and the Optimisation process in Machine Learning (and few illustrations of its benefits)

Gaël Poëtte, David Lugato, Paul Novello

An analogy between solving Partial Differential Equations with Monte-Carlo schemes and the Optimisation process in Machine Learning
(and few illustrations of its benefits)

Gaël Poëtte*, David Lugato*, Paul Novello*

March 28, 2021

Abstract

In this document, we revisit classical Machine Learning (ML) notions and algorithms under the point of view of the numerician, i.e. the one who is interested in the resolution of partial differential equations (PDEs). The document provides an original and illustrated state-of-the-art of ML errors and ML optimisers. The main aim of the document is to help people familiar with the numerical resolution of PDEs understanding how the most classical machine learning (ML) algorithms are built. It also helps understanding their limitations and how they must be used for efficiency. The basic desired properties of ML algorithms are stated and illustrated. An original (PDE based) framework built in order to revisit classical ML algorithms and to design some new ones is suggested, tested and gives interesting results. Several classical ML algorithms are rewritten, reinterpreted in this PDE framework, some original algorithms are built from this same framework. The document highlights and justifies an analogy between ML frameworks (such as TensorFlow, PyTorch, SciKitLearn etc.) and Monte-Carlo (MC) codes used in computational physics: ML frameworks can be viewed as instrumented MC codes solving a parabolic PDE with (well identified) modeling assumptions. Finally, an analogy with transport and diffusion is made: improvements of classical optimisers are highlighted, new optimisers are constructed and applied on simple examples. The results are statistically significative and promising enough for counting the design of new transport based ML algorithms amongst the perspectives of this work.

1 Introduction

In this document, we revisit classical Machine Learning (ML) notions and algorithms under the point of view of the numerician, i.e. the one who is interested in developping numerical schemes to approximate partial differential equations (PDEs). The objectives are three-fold:

- the primary one is to present ML differently, in a way that (we hope) is more adapted to people from a PDE background. Some parts of the document may be seen as an attempt to perform an original and illustrated state-of-the-art of ML algorithms, from the numerician point of view.
- The second objective is to identify numerical tricks commonly used for PDE discretisations which could benefit ML algorithms (this will be put forward throughout the many examples of the document). For example, at first sight, a numerician would be astonished to realise that the optimisation algorithms, despite being non-consistent (see section 3.3), give satisfactory results. The PDE framework we build in this document, in our opinion, helps understanding why and

*CEA, CESTA, DAM, F-33114 Le Barp, France

how. From the next analysis, we show that relatively classical notions from numerical analysis to solve PDEs could benefit ML algorithms (consistency, *a posteriori* errors bounds, stability conditions etc.) or at least their understanding.

– Conversely, ML algorithms are known to intensively make use of vectorised (fine grained HPC) architecture: the analysis could lead to new ideas to make a better use of vectorisation within PDE resolutions. ML algorithms are also known to handle efficiently correlated inputs or treat vectorial outputs (which remain complex in uncertainty quantification for example). In uncertainty analysis, Kriging [71, 5] needs the minimisation of a nonconvex functional and ML algorithms seem to perform well on this point. In brief, what we mean here is that numerical and uncertainty analysis (pilars of the V&V concept [3] in computational physics) could also benefit from the next analysis of ML algorithms.

At the end of the document (section 4), we also hope the analogy between ML algorithms and Monte-Carlo (MC) codes such as the ones implemented and used at the CEA (and in many other industrial contexts) will be clearer. This document does, in no way, represent a first attempt to relate transport models to learning theory: there is a whole section in *Asymptotic Analysis of Transport Processes* [54] entitled *learning theory and other examples*. In this document (section 4), we try to sharpen this link, to better characterise it with respect to what is currently classically used in ML frameworks and in MC resolutions of (stochastic or not) PDEs [54, 10, 9, 14, 39, 6, 60, 59, 57, 55, 73, 12].

The aim of ML can be summed up, without loss of generality, by a will to provide a mapping, an approximation, of an application

$$X \sim \mathrm{d}\mathcal{P}_X \in \mathcal{D}_{in} \subset \mathbb{R}^{d_{in}} \longrightarrow u(X) \in \mathcal{D}_{out} \subset \mathbb{R}^{d_{out}}.$$

This application u transforms an input vector X in dimension d_{in}, following a certain distribution $X \sim \mathrm{d}\mathcal{P}_X$, into an output one $u(X)$. In probability theory, X is called a random vector. Nothing prevents the components of X from being correlated (on the contrary, they are often correlated, for example, for signal or image processing). The latter random vector is transformed into $u(X) \in \mathcal{D}_{out} \subset \mathbb{R}^{d_{out}}$ which is unknown $\forall X \in \mathcal{D}_{in}$. In practice, we only have access to $(u(X_i), w_i)_{i\in\{1,\dots,N\}}$ *via* realisations/snapshots/points of experimental design $(X_i, w_i)_{i\in\{1,\dots,N\}}$. In general, the weights are equals to $w_i = \frac{1}{N}$ but nothing prevents them from being different than this (but in this document, we assume that $\sum_{i=1}^{N} w_i = 1$).

Now, ML supposes looking for an architecture $X \sim \mathrm{d}\mathcal{P}_X \in \mathcal{D}_{in} \times \theta \in \Theta \subseteq \mathbb{R}^{Card(\theta)} \longrightarrow u(X,\theta) \in \mathbb{R}^{d_{out}}$ which will fit $u(X) \in \mathbb{R}^{d_{out}}$ thanks to the fine tuning of the vector of weights $\theta \in \Theta$: this fine tuning will be described (and even, in a sense, revisited, in section 3). Note that the notation for the architecture (Artificial Neural Network ANN) is general: it can denote a Multi Layer Perceptron (MLP), a Deep Neural Network (DNN) or any other kind of approximation (Convolutional, ResNet etc.)[1]. The notation $u(X,\theta)$ is concise but abusive: in order to completely describe an architecture, one should provide *at least* a number of neurons n_l in layer l, a number of layers N_l and one (or several) activation function(s) σ (for a fully connected NN). The previous (hyper)parameters will be recalled in the text when relevant but not in the notations $u(X,\theta)$ for the sake of conciseness (and, we insist, without loss of generality in this document).

The architectures classically used in ML have several important properties:

– the number of parameters $Card(\theta)$ in vector θ depends on N_l and n_l: suppose there are n_c neurons in layer c and N_l layers, then $Card(\theta) = \sum_{c=1}^{N_l}(n_{c-1}+1)n_c + n_{c+1}n_c$[2]. It consequently grows *polynomially* with the dimension, the number of layers/neurons and linearly with the number of outputs. To give an idea, well-known and commonly used methods such as polynomial regression

[1] As every type of architecture resumes to the tuning of their weights even if not having the same power of expressiveness, denoted by ϵ later on.

[2] if $n_0 = d_{in}, n_{N_l} = d_{out}$.

(which ensures spectral convergence [57, 21, 7]) have a complexity which grows exponentially with the dimension and the truncation order and linearly with the number of output, see [60, 58]. ANN may represent an interesting alternative to polynomial regression or Kriging etc. in moderate to high dimensions.

- Futhermore, once the best $\theta \in \Theta$ obtained, the cost of an execution of the architecture is independent of the number of training points N. This is not the case for Kriging (or kernel based methods) for example, see [58, 60, 71, 5]. Hence their relevance in a big data context.
- With low regularity assumptions on u (such as $u \in L^2$, of importance in our applications), relatively general assumptions on the activation functions σ and assuming we can find the relevant $\theta \in \Theta$, the ANN approximation $u(X, \theta)$ converges toward u as $Card(\theta) \to \infty$, see [30, 2, 43] for example.

According to the above properties, this kind of approximation is particularly well fitted to important correlated dimensions in a big data context. In the next section, we explain in which way $u(X, \theta)$ can approximate $u(X)$.

2 The different types of errors in Machine Learning

In order to fit the parameters, the idea is to look for the set of weights $\theta \in \Theta$ minimising a functional

$$J(\theta) = \int L\left(u(X), u(X, \theta)\right) \mathrm{d}\mathcal{P}_X.$$

Functional $(x, y) \in \mathbb{R}^2 \longrightarrow L(x, y)$ is called the loss function (see example 2.0.1).

Different theorems (Hornik [30], Barron [2], Lu-Su *et al.* [43]) ensure that some architectures $u(X, \theta)$ are converging as the number of parameters[3] $Card(\theta) \longrightarrow \infty$ for a given choice of loss function L. Converging with $Card(\theta)$ or not, we can always decompose $u(X)$ as

$$u(X) = u(X, \theta^*) + \epsilon(X),$$

where

- $u(X, \theta^*)$ corresponds to the best approximation with architecture $u(X, \theta)$,
- $\epsilon(X)$ is the residue of $u(X)$.

With the previous decomposition, we have

$$J(\theta) = \int L(u(X, \theta^*) + \epsilon(X), u(X, \theta))\, \mathrm{d}\mathcal{P}_X.$$

Now, in practice, we do not have access to J but rather to an approximation of J based on the experimental design $(X_i, w_i)_{i \in \{1, \dots, N\}}$. This experimental design (or snapshot or training set) is nothing more than a set of points discretising $(X, \mathrm{d}\mathcal{P}_X)$ in the sense that $\forall f \in L^2(\mathcal{D}_{in}, \mathrm{d}\mathcal{P}_X)$

$$\sum_{i=1}^{N} w_i f(X_i) \underset{N \to \infty}{\overset{\mathcal{L}}{\longrightarrow}} \int f(X)\, \mathrm{d}\mathcal{P}_X.$$

The convergence above is in law ($\mathcal{L}$) for an MC experimental design but may be in L^1/L^2 depending on the set of points and the regularity of the integrand. The convergence rate depends on the kind of chosen training set/experimental design[4].

[3] $Card(\theta)$ is related to both the width of the architecture (with the number of neurons) or its depth (with the number of layers).

[4] It can be $\mathcal{O}(\frac{1}{\sqrt{N}})$ for MC, $\mathcal{O}(\frac{1}{N})$ for uniform designs, faster for Gauss points, etc., see [23, 60] for example.

Let us introduce

$$J_N(\theta) = \sum_{i=1}^{N} w_i L(u(X_i), u(X_i, \theta)),$$

and the discrete probability measure

$$\mathrm{d}\mathcal{P}_X^N(X) = \sum_{i=1}^{N} w_i \delta_{X_i}(X).$$

We can rewrite J_N as

$$J_N(\theta) = \int L(u(X), u(X, \theta))\, \mathrm{d}\mathcal{P}_X^N,$$

and J as

$$\begin{aligned} J(\theta) &= \int L(u(X, \theta^*) + \epsilon(X), u(X, \theta))\, \mathrm{d}\mathcal{P}_X - J_N(\theta) + J_N(\theta), \\ &= \underbrace{\int L(u(X, \theta^*) + \epsilon(X), u(X, \theta)) \underbrace{\left[\mathrm{d}\mathcal{P}_X - \mathrm{d}\mathcal{P}_X^N\right]}_{\overline{\mathrm{d}\mathcal{P}_X^N}}}_{\text{generalisation error } \overline{J}(\theta)} + J_N(\theta). \end{aligned} \tag{1}$$

Above, J is the sum of what is called the generalisation error/gap $\overline{J}$ and J_N. Of course, J_N is what is used in practice in order to tune θ. The generalisation gap depends on points outside the training set, see the definition of $\overline{\mathrm{d}\mathcal{P}_X^N}$. Note that $\overline{\mathrm{d}\mathcal{P}_X^N}$ is a measure, but not a probability measure (as it is not necessarily positive nor does not necessarily sums up to 1), which tends to the null measure as N grows. The set of points used in order to estimate the generalisation gap $\overline{J}$ is commonly called the *test-set* in a cross-validation context. Assume $\hat{\theta}$ is the set of weights obtained after the optimisation step on J_N. Then the *a posteriori* estimation of $\overline{J}(\hat{\theta})$, if accurate enough[5], provides an estimation of the error made by architecture $u(X, \hat{\theta})$ to approximate $u(X)$. Functional J can also be decomposed into

$$J(\theta) = \overline{J}(\theta) + \underbrace{J_N(\theta) - J_N(\theta_N^*)}_{\text{optimiser error}} + J_N(\theta_N^*), \tag{2}$$

Where θ_N^* is the minimiser of $J_N(\theta)$. The above simple expression (2) emphasizes the fact that even if we estimate accurately $\overline{J}$, estimating the optimiser error is not easy without knowing $J_N(\theta_N^*)$. For this reason, in the next sections in which we focus on the optimiser error, we consider test-cases for which we know θ_N^* in order to avoid misleading interpretations. Because misleading interpretations are possible: $J(\theta)$ is a mix of errors. To give a better idea, let us rewrite (2) as

$$J(\theta) = \underbrace{J(\theta^*)}_{\text{truncation error}} + \underbrace{\overline{J}(\theta)}_{\text{generalisation error}} + \underbrace{J_N(\theta_N^*) - J(\theta^*)}_{\text{integration error}} + \underbrace{J_N(\theta) - J_N(\theta_N^*)}_{\text{optimiser error}}. \tag{3}$$

In the above expression (3), there are four types of errors:

– the truncation error expresses the error related to the choice of the architecture as

$$J(\theta^*) = \int L(u(X, \theta^*) + \epsilon(X), u(X, \theta^*))\, \mathrm{d}\mathcal{P}_X = \int L(\epsilon(X), 0)\, \mathrm{d}\mathcal{P}_X.$$

It corresponds to what can be asymptotically reached

[5] i.e. if there are enough points in the *test set* to accurately evaluate it.

- as $N \gg 1$,
- and with an infinitely efficient optimisation of the weights θ.

For a converging architecture, this error goes to zero as $Card(\theta) \to \infty$, see [30, 2, 43].

- The optimiser error is positive (as $J_N(\theta_N^*) \leq J_N(\theta), \forall \theta$ by definition) and expresses the error related to the fact that the optimisation algorithm (see next section) may not exactly find θ_N^*. Still, this does not mean that we will have a bad model $u(X, \theta)$ at the end of the optimisation step:
  - for example, the optimisation algorithm can get close to θ^* when looking for θ_N^*[6].
  - Or suppose that the optimiser finds $\hat{\theta}$. Even if J_N and J are far from each other (important integration error) so that θ^* is not close to $\theta_N^* = \hat{\theta}$ or if the optimiser is not good enough and $\hat{\theta}$ is far from θ_N^*, this does not mean we will have a bad model $u(X, \hat{\theta})$. We have

$$J(\theta^*) - J(\hat{\theta}) = (\theta^* - \hat{\theta}) \nabla_\theta J(\hat{\theta}) + \mathcal{O}((\theta^* - \hat{\theta})^2),$$

    and if $\nabla_\theta J(\hat{\theta}) \ll 1$, then, intuitively, $J(\theta^*) \approx J(\hat{\theta})$. The gradient of J strongly depends on the choice of the architecture (number of layers, of neurons, activation function choice etc.) and of the loss function. In a sense, having small gradients for $\theta \to J(\theta)$ here seems to lead to robust models/architectures. Unfortunately, small gradients are also known to lead to longer optimisation/learning times, see [28].
  - We have, to our knowledge, no idea of how to estimate the optimiser error. The PDE framework we introduce in section 3.5 may help on this topic.

Remark 2.1 *The optimiser error is neither a part of the training error J_N nor of the generalisation error $\overline{J}$. It can not easily be estimated or accounted for by computing $J_N(\hat{\theta})$ and $\overline{J}(\hat{\theta})$ where $\hat{\theta}$ is the set of weights selected after a training phase. For this reason, when studied in the next sections of this paper, we have to rely on benchmarks for which we* a priori *know the local minima. It justifies the simplicity of the test-cases of this document and allows avoid misleading conclusions.*

- The integration error depends on both
  - how fast J_N converges toward J,
  - how close θ_N^* is from θ^* even if J_N remains quite far from J.

  This error is unsigned (independently of the choice of L) and can consequently be misleading: $J_N(\theta_N^*) - J(\theta^*)$ can compensate the other errors. Practically, this means that an approximation[7] $u(X, \hat{\theta}_N)$ with less points than an approximation[8] $u(X, \hat{\theta}_{N'})$ with $N \ll N'$ can yield better performances.
- Of course, if all these errors are small, then we may be able to solve the problem we want to solve, namely (minimising J by) minimising J_N.

In every case, decomposition (1) states that once $\hat{\theta}$ obtained after a training step with N points, the quality of our approximation $u(X, \hat{\theta})$ can be estimated

- by evaluating $\overline{J}(\hat{\theta})$ thanks to a test set,

[6] and sometimes hopefully miss it! Indeed, there are many algorithms, such as dropout [69] for example, which tries to fit θ but modify the results of the training to estimate the generalisation gap, see section 3.6.2.

[7] obtained thanks to an optimiser working on J_N and outputting $\hat{\theta}_N$.

[8] obtained thanks to an optimiser working on $J_{N'}$ and outputting $\hat{\theta}_{N'}$.

– and $J_N(\hat{\theta})$ on the training set,

– and summing up both contributions to obtain $J(\hat{\theta}) = \overline{J}(\hat{\theta}) + J_N(\hat{\theta})$.

But the above discussion shows that if $J(\hat{\theta})$ does not reach our expectations in term of accuracy, then it will be hard identifying the main contributor of the error. The work described in [50] may represent an attempt to do it.

Example 2.0.1 (An example of function $J(\theta)$) *We here suggest presenting an example of function that we may want to minimise*

$$\theta \in \mathbb{R} \longrightarrow J(\theta) = -e^{-\frac{\theta^2}{4}} - 2e^{-\frac{(\theta-6)^2}{4}}. \tag{4}$$

The function has two minima, at $\theta = 0$ and $\theta = 6$ and $J(6) < J(0)$. They are known, see remark 2.1. Function (4) could, for example, be the loss function $J(\theta) = \int L(u(X), u(X,\theta))\, \mathrm{d}\mathcal{P}_X$ of an architecture with

– *loss function $x, y \to L(x,y) = (x-y)$ (L is not necessarily related to a norm),*

– *to approximate $X \sim \mathrm{d}\mathcal{P}_X \to u(X) = 0$,*

– *with activation function $x, \theta \to e^{-\frac{(x-\theta)^2}{4}} + 2e^{-\frac{(-6-(x-\theta))^2}{4}}$ (θ is only a bias here, see [28]),*

– *only one point in the training set $(X_1 = 0, u(X_1) = 0)$,*

– *only one layer and one neuron so that the architecture is $u(x,\theta) = e^{-\frac{(x-\theta)^2}{4}} + 2e^{-\frac{(-6-(x-\theta))^2}{4}}$.*

Note that, equivalently, it can be the loss function of an architecture with

– *loss function $x, y \to L(x,y) = (x-y)$,*

– *to approximate $X \sim \mathrm{d}\mathcal{P}_X \to u(X) = 0$,*

– *with activation function $x, \theta \to e^{-\frac{(x-\theta)^2}{4}}$ (θ is only a bias here, see [28]),*

– *only two points in the training set $\{(X_1 = 0, u(X_1) = 0), (X_2 = 6, u(X_2) = 0)\}$,*

– *only one layer and two neurons with the first one with weight $w_1 = 1$ and the second with $w_2 = 2$,*

– *and the two bias $\theta_1 = \theta_2 = \theta$.*

With this simple example, we can see that two different architectures can lead to the same optimisation problem for some particular directions of the space of the weights.

This simple problem will constitute a fil rouge *which will be revisited all along this document to better understand the behaviour of ML algorithms.*

With the above expression, we realise that we may have to be lucky to obtain a good θ and/or to identify where the error comes from. Fortunately, from the convergence theorems [30, 2, 43], we know that we can make the terms depending on ϵ arbitrarily small by refining (increasing the width/depth such that $Card(\theta) \to \infty$) the architectures. This means that once an architecture is chosen, ϵ is fixed, only three errors remain:

– the generalisation error: it is *a posteriori* estimated and helps attesting of the quality of approximation $u(X,\theta)$.

– The integration error: it strongly depends on the choices of the points $(X_i, w_i)_{i\in\{1,\dots,N\}}$. In [51, 52], the authors use the distribution of points as a lever in order to improve the performances of an architecture by decreasing this error.

– The optimiser error: in this document, we focus on this error and on the optimisation of θ mainly because several goal-oriented sensitivity analysis [50] pointed out that the optimiser (and its parameters) is very often the main contributor to having small global errors.

Let us study the optimiser error *via* the study of the optimisation algorithms at hand.

3 The optimisation step

In this section, we present the optimisation step, i.e. how θ is found in practice. Note that this presentation is, to our knowledge, original and different from what can be found in classical ML books [28]. This presentation is adressed to people familiar with PDE resolution/discretisation and numerical analysis. It aims at putting forward an analogy between ML algorithms and MC codes (for neutronics, photonics, plasma physics, economics or finance, see for examples [10, 9, 14, 39, 6, 12, 60]). It also allows

– going through several interesting ML algorithms (GD, SGD, dropout, etc.),

– candidely presenting why and how they are used,

– putting them in a common framework in order to ease their analysis and understanding (especially for the ones which are not familiar with the furnished ML literature).

Let us begin by a presentation of what is commonly called the forward and backward propagations.

3.1 The forward and backward propagations

As explained in the previous lines, we aim at minimising

$$J(\theta) = \int L(u(X), u(X,\theta))\,\mathrm{d}\mathcal{P}_X = \int L(X,\theta)\,\mathrm{d}\mathcal{P}_X.$$

We consequently have to look for θ cancelling the gradient

$$F(\theta) = \nabla_\theta J(\theta) = \int L'(u(X), u(X,\theta))\nabla_\theta u(X,\theta)\,\mathrm{d}\mathcal{P}_X = \int L'(X,\theta)\nabla_\theta u(X,\theta)\,\mathrm{d}\mathcal{P}_X,$$

where $L'(x,y) = \partial_y L(x,y)$. Note that by cancelling the gradient, we may unintentionally also look for maxima of J, which are not of interest. The forward propagation consists in evaluating $J(\theta)$ for a given θ. The backward propagation consists in evaluating $\nabla_\theta J(\theta)$ in order to correct θ.

3.1.1 From deterministic optimisation algorithms...

Looking for minima of J implies looking for θ cancelling $\nabla_\theta J(\theta)$ and making sure that θ cancelling $\nabla_\theta J(\theta)$ is not a maximum or a saddle point. The first reflex as a numerician is to consider the most classical optimisation algorithm to cancel F, the Newton algorithm:

– it is based on the Taylor development

$$F(\theta + \Delta\theta) = F(\theta) + \nabla_\theta F(\theta)\Delta\theta + \mathcal{O}(\Delta\theta^2), \tag{5}$$

– in which assuming local linearity (i.e. $\mathcal{O}(\Delta\theta^2) \ll 1$),

– leads to $F(\theta^*) = 0$ if $\Delta\theta = \theta^* - \theta$.

As consequence, in practice, one looks for θ^* such that

$$0 = \nabla_\theta J(\theta) + \nabla_{\theta,\theta} J(\theta)(\theta^* - \theta). \tag{6}$$

In general, F is not linear and we just described one step of an iterative algorithm.

Of course, θ^* is the unknown: we have

$$\theta^* = \theta - \left[\nabla^2_{\theta,\theta} J(\theta)\right]^{-1} \nabla_\theta J(\theta).$$

We consequently need to inverse the Hessian $\nabla^2_{\theta,\theta} J(\theta)$. Now, one has to keep in mind that

1. it is of size $Card(\theta) \times Card(\theta) \gg 1$: its inversion is not affordable in practice due to its size.
2. Functional J is not strictly convex which means that the eigenvalues of $\nabla^2_{\theta,\theta} J(\theta)$ are not all strictly positive. Typically, this means that the Newton algorithm may 'go uphill' or remain stuck on a saddle point [19] (or see example 3.1.1). In practice, in order to avoid this situation, it is common practice to replace $\nabla^2_{\theta,\theta} J(\theta)$ by (an approximation of) $|\nabla^2_{\theta,\theta} J(\theta)|$, see [19] for example, where $|H|$ has the same eigenvectors as H and the absolute values of the eigenvalues of H as eigenvalues. In the following, $|H|$ is called the *absolute value of* H. As a consequence, through the iterations, (6) is replaced by

   $$0 = \nabla_\theta J(\theta) + |\nabla_{\theta,\theta} J(\theta)|(\theta^* - \theta). \tag{7}$$

   Using the expression above instead of the (6) ensures going in the opposite direction of the gradient $\nabla_\theta J$ even if $\nabla^2_{\theta,\theta} J$ is not strictly positive (see example 3.1.1).
3. Even once (6) replaced by (7), the non-convexity of the functional to minimise implies a problem of non unicity of the minimum (of θ cancelling $\nabla_\theta J$). In order to avoid remaining stuck in a local minimum, one needs to run several times the Newton algorithm at several starting points, see example 3.1.1.

Let us tackle every of the above points in a pedagogical example based on the minimisation of (4) of example 2.0.1.

Example 3.1.1 (Newton for a nonconvex functional) *We here suggest presenting the behaviour of a Newton algorithm (cf. (6)) and a modified one (according to (7)) in order to minimise the nonconvex function (4) of example 2.0.1. Applying a Newton algorithm is easy here because $d_{in} = 1$ and J'' is easy to compute so that only point 1 above is simplified (i.e. the matrix inversion). Equations (6) and (7) resume to*

$$\theta_{n+1} = \theta_n - \frac{J'(\theta_n)}{J''(\theta_n)} \text{ and } \theta_{n+1} = \theta_n - \frac{J'(\theta_n)}{|J''(\theta_n)|}.$$

Figure 1 presents the results obtained with the two Newton algorithms. The two top pictures are obtained with (6), the four bottom ones with (7). The left column displays the functional $\theta \to J(\theta)$ given by (4) together with the serie of coordinates $((\theta_n, J(\theta_n))_{n\in\{1,\dots,n_{\text{epoch}}=2000\}}$ through $n_{\text{epoch}} = 2000$ iterations/epochs. The best point $(\theta_{n_{best}}, J(\theta_{n_{best}}))$ is in blue and the last point $(\theta_{n_{last}}, J(\theta_{n_{last}}))$ is in magenta. The right column presents the straight lines $\theta^* = 0$ and $\theta^* = 6$ together with the values of $n \to \theta_n$ obtained by the concerned algorithms.

On the top pictures, we can see that a classical Newton (6) is not adapted for a nonconvex functional: the process $(\theta_n)_{n\in\{1,\dots,n_{\text{epoch}}=2000\}}$ goes 'uphill' and does not follow the opposite direction of the gradient. On the right picture, we see that it does not tend toward neither $\theta^* = 0$ nor $\theta^* = 6$.

On another hand, on the bottom pictures, the modified Newton algorithm (7) allows converging toward $\theta^* = 0$ or $\theta^* = 6$. It converges toward $\theta^* = 0$ or $\theta^* = 6$ depending on where the Newton has been initialized. On the right pictures, we can even see that the convergence is fast: $\theta^* = 0$ or $\theta^* = 6$ are reached within about 10 iterations. Note that with this algorithm, $(\theta_{n_{best}}, J(\theta_{n_{best}}))$=$(\theta_{n_{last}}, J(\theta_{n_{last}}))$: the last point is always the best point amongst the iterations.

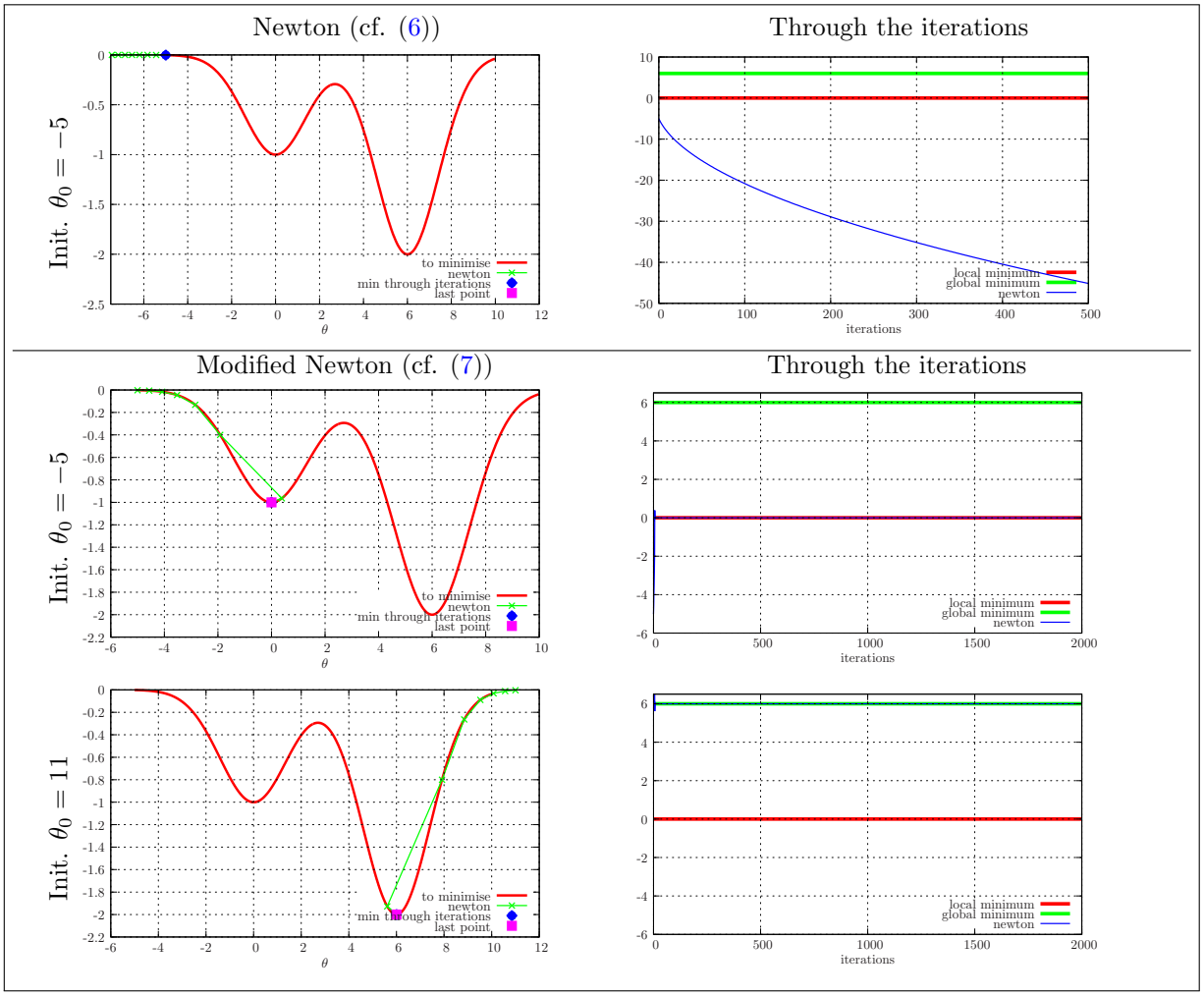

Figure 1: Top: results obtained with a classical Newton algorithm (6) on the nonconvex functional (4). Bottom: results obtained with the modified Newton algorithm (7) on the same functional for several initialisations. Comments are provided in example 3.1.1.

As we will aim at minimising some non-convex functionals in the following sections, we will keep in mind the fact that care must be taken to have an algorithm which *follows the opposite direction of the gradient* together with *relying on multiple initialisations* (see example 3.1.1). Concerning this second point, closely related to the existence of several local minima, many authors studied the number of local minimum to be adressed: in [4], the authors show that even *with only one layer, one neuron,* $x \in \mathbb{R}^{d_{in}}$ *and if* $L(\sigma(\theta \cdot x), y)$ *is continuous and bounded (independently of the choice of the loss function*

L and of the activation function σ), then $\theta \to L(\sigma(\theta \cdot x), y)$ has exponentially many local minima with N and d_{in}. In other words, *such (deterministic) (modified) Newton algorithm may need a number of initialisations growing exponentially fast with N and d_{in}* in order to find/deal with the many local minima. Example 3.1.2, presenting more quantitative results, aims at quantifying the probability for the optimiser (here, the Newton or the modified Newton) to reach the global or a local minimum.

Example 3.1.2 (Newton for a nonconvex functional, quantitative results) *The results of example 3.1.1 related to figure 1 were mainly qualitative and helped understanding and illustrating what may happen in practice. We here suggest some quantitative ones. This example is motivated by the fact that the results of the top line of figure 1 are not really representative of the general behaviour of the Newton algorithm (based on (6)). Indeed, if initialised elsewhere, the same algorithm could have lead to finding a global/local minimum. For example, if θ_0 is such that $J''(\theta_0) > 0$, then the Newton algorithm will converge toward a local minimum. In the above example, with J given by (4), it is easy characterising the regions where J'' is positive or negative:*

$$\begin{array}{l} J''(\theta) \leq 0 \text{ for } \theta \in [-\infty, -1.414125] \cup [1.32005, 4.613025] \cup [7.4145, +\infty], \\ J''(\theta) > 0 \text{ for } \theta \in [-1.414125, 1.32005] \cup [4.613025, 7.4145]. \end{array} \tag{8}$$

In order to find the global minimum amongst the potentially many local ones [4], it is common having resort to multiple initialisations. In this document, we say that we change *the seed, or rely on N_{seed} different seeds/initialisations. Practically, it consists in sampling θ_0 according to a prescribed probability distribution. This distribution is often called the* initialiser *in an ML context. In this document, we mainly consider initialiser $\theta_0 \sim \mathcal{U}([-15, 15])$: the guess θ_0 is sampled uniformly within interval $[-15, 15]$. Of course, other initial distributions can be used (see table 1 and the related discussion).*

For each initialisation/seed, we run a Newton based on (6). Let us denote by $\theta_{\min}$ the parameter $\theta_{\min} \in (\theta_n)_{n \in \{1,...,n_{\text{epoch}}=2000\}}$ such that $J(\theta_{\min}) \leq J(\theta_n), \forall n \in \{1, ..., n_{\text{epoch}} = 2000\}$. Then,

- *the probability of having $\theta_{\min}$ in the vicinity of $\theta^* = 0$ is $\approx 9.1\%$,*
- *the probability of having $\theta_{\min}$ in the vicinity of $\theta^* = 6$ is $\approx 9.3\%$,*
- *the probability of going uphill and not being in the above vicinities is $\approx 81.6\%$.*

Table 1 sums-up those probabilities together with others, obtained for different initialisers:

- *$\theta_0 \sim \mathcal{U}([-15, 15])$, i.e. an uniform sampling in interval $[-15, 15]$,*
- *$\theta_0 \sim \mathcal{G}(0, 1)$, i.e. a gaussian sampling of mean 0 and variance 1,*
- *$\theta_0 \sim \mathcal{G}(3, \frac{3}{2})$, i.e. a gaussian sampling of mean 3 and variance $\frac{3}{2}$.*

Expectedly, the probabilities are sensitive to the initial distribution of θ_0: an attempt will be made later on to understand how to build better initialisers (see example 3.6.3 and the related discussion about the solution of the Fokker-Planck equation). In the next examples of this document, we will mainly focus on initialiser $\theta_0 \sim \mathcal{U}([-15, 15])$.

Note that we must define more precisely the term vicinities*: in this example, the vicinities are defined by an interval $[\theta^* - \epsilon, \theta^* + \epsilon]$ for both $\theta^* = 0$ and $\theta^* = 6$ where ϵ remains to be chosen. In practice, we choose $\epsilon = 10^{-2}$ (even if for the Newton algorithms, ϵ is close to the threshold of the stopping criterion of the Newton algorithms, i.e. 10^{-14}).*

In the same conditions as above, if we apply a modified Newton algorithm based on (7), then

- *the probability of having $\theta_{\min}$ in the vicinity of $\theta^* = 0$ is $\approx \frac{2.7028--15}{30} = 59\%$,*

– *the probability of having* $\theta_{\min}$ *in the vicinity of* $\theta^* = 6$ *is* $\approx \frac{15-2.7028}{30} = 41\%$,

– *the probability of going uphill is* 0%.

The previous probabilities are recalled in table 1 for several initialisers for θ_0 *and for the Newton and the modified Newton algorithms. This table will be useful to compare the performances of other algorithms later on. The terms, notations and parameter choices defined in this example are common to all the next quantitative studies, it will ease the next algorithm comparisons.*

	Vicinity of $\theta^* = 0$	Vicinity of $\theta^* = 6$
Initialiser $\theta_0 \sim \mathcal{U}([-15, 15])$	$\mathbb{P}(\theta_{\min} \in [-10^{-2}, 10^{-2}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-2}, 6 + 10^{-2}])$
Newton based on (6)	9.10%	9.30%
modified Newton based on (7)	59.0%	41.0%
Initialiser $\theta_0 \sim \mathcal{G}(0, 1)$	$\mathbb{P}(\theta_{\min} \in [-10^{-2}, 10^{-2}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-2}, 6 + 10^{-2}])$
Newton based on (6)	82.8%	1.95×10^{-6}%
modified Newton based on (7)	99.6%	0.40%
Initialiser $\theta_0 \sim \mathcal{G}(3, \frac{3}{2})$	$\mathbb{P}(\theta_{\min} \in [-10^{-2}, 10^{-2}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-2}, 6 + 10^{-2}])$
Newton based on (6)	12.9%	13.9%
modified Newton based on (7)	42.1%	57.9%

Table 1: Probabilities for the Newton (6) and modified Newton (7) algorithm to recover the local ($\theta^* = 0$) or the global ($\theta^* = 6$) minimum of (4) for different distributions of θ_0. Comments are provided in example 3.1.2.

As explained above, examples 3.1.1 and 3.1.2 do not tackle the hessian inversion problem (described in point 1 above) as the examples are in a low dimensional context. The hessian and the absolute value of the hessian computations here are immediate. In the next example, we present the Gradient Descent algorithm which is classically used when the inverse of the (absolute value of the) hessian is not at hand.

Example 3.1.3 (Newton for a nonconvex functional without hessian inversion)) *In this example, we suppose we cannot have access to the inverse of the (absolute value of the) hessian. Gradient Descent suggests arbitrarily choosing a value* $\gamma > 0$*, often called the learning rate, in order to roughly approximate it. As a consequence, a Gradient Descent (GD) algorithm (classical if* J'' *can not be computed) is such that*

$$\theta_{n+1} = \theta_n - \frac{1}{|J''(\theta_n)|} J'(\theta_n) \approx \theta_n - \gamma J'(\theta_n). \tag{9}$$

Figure 2 revisits the results of example 3.1.1 and figure 1 with a GD algorithm for several learning rates $\gamma = 0.05, 3.5, 25$*. The initialisation points are exactly the same, only the descent algorithm is changed. The top pictures shows the results obtained from the starting point* $\theta_0 = -5$*, with learning rate* $\gamma = 0.05$*: the left column displays the functional* $\theta \to J(\theta)$ *given by (4) together with the serie of coordinates* $((\theta_n, J(\theta_n))_{n \in \{1,\dots,n_{\text{epoch}}=2000\}}$*. The best point* $(\theta_{n_{best}}, J(\theta_{n_{best}}))$ *is in blue and the last point* $(\theta_{n_{last}}, J(\theta_{n_{last}}))$ *is in magenta. The blue and magenta points match: the algorithm stops at the best point in this case. For such low learning rate (relative to the slope of the function to minimise), the GD algorithm stops just before some major improvements. The right column presents the straight lines* $\theta^* = 0$ *and* $\theta^* = 6$ *together with the values of* $n \to \theta_n$*: we*

can see that the algorithm would need more than $n_{\text{epoch}} = 2000$ *iterations to attain the vicinity of* $\theta^* = 0$. *The pictures from the second line present the results obtained from starting point* $\theta_0 = -5$ *and* $\gamma = 3.5$. *This time, there are enough iterations/epochs for the GD algorithm to stagnate in the vicinity of* $\theta^* = 0$. *Still, see the right picture of the second line, it needs more iterations than the modified Newton algorithm of figure 1 in the same conditions. The pictures from the third line present the results obtained from the* $\theta_0 = 11$ *starting point and* $\gamma = 3.5$. *In this case, the last point is not the best point: in this sense, GD is not an optimisation algorithm as it does not ensure cancelling the gradient. It is more an exploration algorithm: this means that one must monitor the error and keep in memory the point with the lowest value in order to avoid losing it. In other words, while tuning your weights* θ, *you must be able to instrument the descent, monitor the error and keep track of the best weights along the iterations/epochs. Furthermore, with this choice of* $\gamma = 3.5$, *the algorithm oscillates between several states and does not go below a certain value. Note that it is common practice to decrease the learning rate after a certain number of arbitrary iterations/epochs but, to our knowledge, the learning rate reduction strategies are rules of thumbs. The last line of figure 2 tackles the case* $\theta_0 = 11$ *and* $\gamma = 25.0$*: the GD algorithm almost immediately jumps outside the region of interest and lands in a zero-gradient region where it remains stuck.*

Example 3.1.4 (Nonconvex functional without hessian inversion, quantitative results) *The results of example 3.1.3 related to figure 2 were mainly qualitative. We here suggest some quantitative ones, just as in example 3.1.2.*

Let us consider initialiser $\theta_0 \sim \mathcal{U}([-15, 15])$ *as in example 3.1.2. For each starting points, a GD based on (9) is run. Let us denote by* $\theta_{\min}$ *the parameter* $\theta_{\min} \in (\theta_n)_{n\in\{1,\dots,n_{\text{epoch}}=2000\}}$ *such that* $J(\theta_{\min}) \leq J(\theta_n), \forall n \in \{1, ..., n_{\text{epoch}} = 2000\}$ *for the concerned run. The probabilities of recovering the local* $\theta^* = 0$ *or the global* $\theta^* = 6$ *minimum of (4) with GD are displayed in table 2 for several values of the learning rate* γ.

For a small learning rate $\gamma = 0.05$, *GD gives a* 25% *probability of having* $\theta_{\min}$ *in the vicinity of the local minimum* $\theta^* = 0$ *and a* 28% *one for* $\theta_{\min}$ *to be in the vicinity of the global minimum* $\theta^* = 6$. *The probability of not reaching the interesting vicinities is* $100 - 25.4 - 28.7 = 45.9\%$ *and is relatively important.*

With $\gamma = 3.5$, *the probability of recovering* $\theta_{\min} \approx 0$ *is higher but the probability of reaching the vicinity of the global minimum is way smaller* ($\approx 0.22\%$) *than for* $\gamma = 0.05$. *This poor performance is of course closely related to the bad behaviour of GD depicted in figure 2 (third line). Here, the probability of not reaching a vicinity of interest is given by* $100-31.2-0.22 = 68.58\%$ *and is greater than for* $\gamma = 0.05$. *For a small learning rate, the probability of stopping in the middle of a slope as in figure 2 (top line) is of course important but is circumvented by the multiple initialisations strategy.*

With $\gamma = 25.0$, *the probabilities of recovering* $\theta_{\min} \approx 0$ *or* $\theta_{\min} \approx 6$ *drastically drop. This poor performance is of course closely related to the bad behaviour of GD depicted in figure 2 (fourth line).*

The previous probabilities, three first lines of table 2, were computed for vicinities defined by intervals of the form $[\theta^* - \epsilon, \theta^* + \epsilon]$ *for* $\theta^* = 0, \theta^* = 6$ *and* $\epsilon = 10^{-2}$. *The last line of table 2 presents the results for GD with* $\gamma = 0.05$ *(giving the previous best performances) for narrower vicinities (i.e.* $\epsilon = 10^{-4}$*). The probabilities for GD to recover these narrow vicinities are certainly extremely small (not statistically significative enough with only* $N_{seed} = 10^4$, *cf. the* 0.00% *line).*

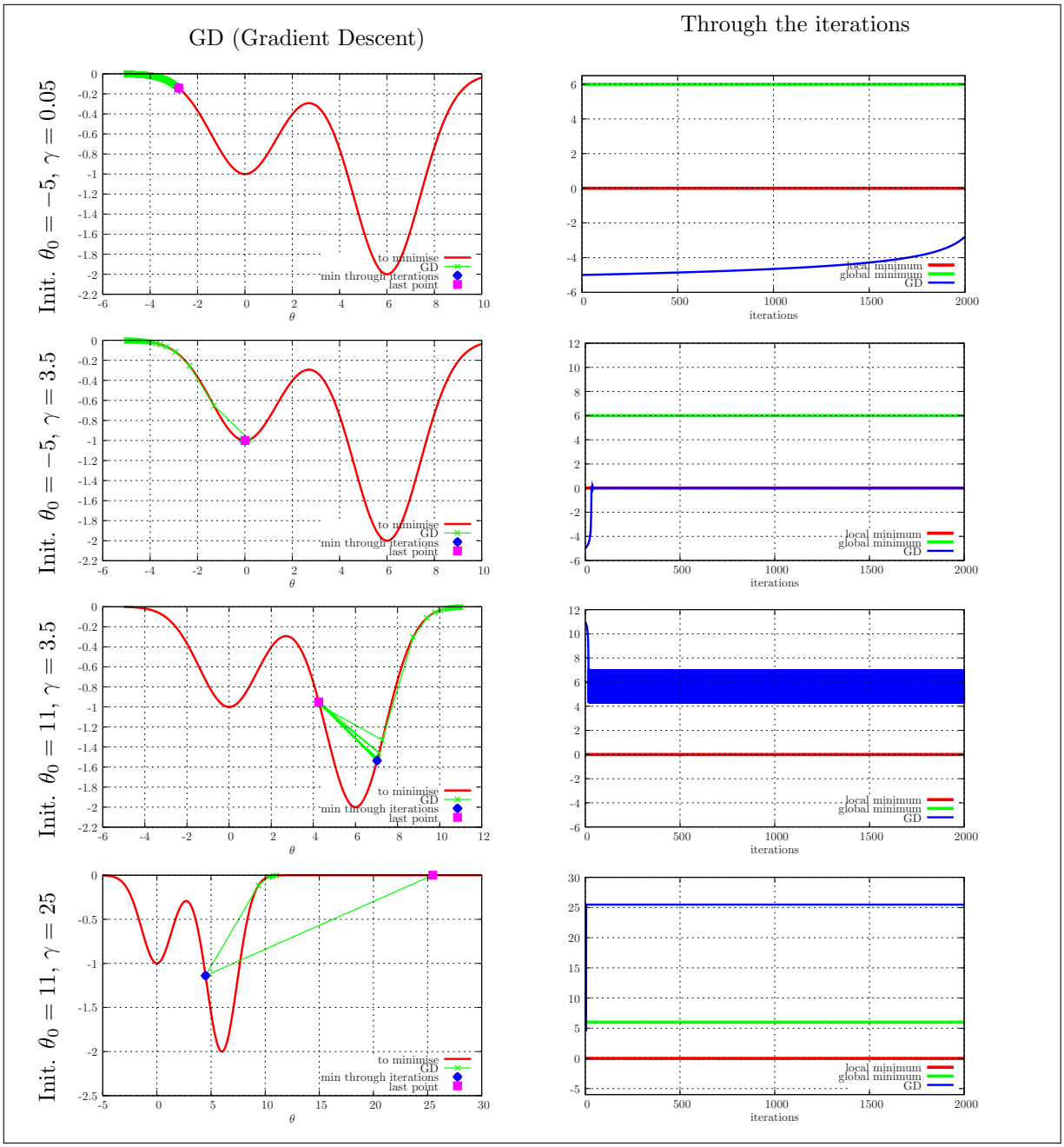

Figure 2: These figures revisit the test-case of example 3.1.1 but with a GD algorithm with learning rates $\gamma = 0.05, 3.5, 25.0$.

Example 3.1.5 (The regularisation effect) *From the last line of figure 2 (obtained with a too important learning rate $\gamma = 25.0$), it is tempting having resort to regularisation technics. Figure 3 investigates the effect of an L^2-regularisation (but any regularisation could be investigated in the same manner). Regularisation technics consist in adding a term to the nonconvex functional we*

Initialiser $\theta_0 \sim \mathcal{U}([-15, 15])$	Vicinity of $\theta^* = 0$	Vicinity of $\theta^* = 6$
GD ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10^{-2}, 10^{-2}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-2}, 6 + 10^{-2}])$
$\gamma = 0.05$	$25.4\% \in [24.59\%, 26.30\%]$	$28.7\% \in [27.75\%, 29.52\%]$
$\gamma = 3.50$	$31.2\% \in [30.33\%, 32.14\%]$	$0.22\% \in [0.120\%, 0.310\%]$
$\gamma = 25.0$	$0.14\% \in [0.065\%, 0.214\%]$	$0.29\% \in [0.180\%, 0.400\%]$
GD ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10^{-4}, 10^{-4}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-4}, 6 + 10^{-4}])$
$\gamma = 0.05$	$0.00\% \in [0.000\%, 0.000\%]$	$0.00\% \in [0.000\%, 0.000\%]$

Table 2: Probabilities for the Gradient Descent (GD) algorithm to recover the local ($\theta^* = 0$) or the global ($\theta^* = 6$) minimum of (4) for several values of the learning rate γ. To compute the probabilities, we have resort to $N_{seed} = 10^4$ multiple initialisations with $\theta_0 \sim \mathcal{U}([-15, 15])$. Confidence intervals for the results are provided. The 0.00% are not significative but attest for a small probability. Comments are provided in example 3.1.4.

aim at minimising. For an L^2 regularisation, (4) is typically replaced with

$$\theta \in \mathbb{R} \longrightarrow J(\theta) = -e^{-\frac{\theta^2}{4}} - 2e^{-\frac{(\theta-6)^2}{4}} + \xi(\theta - \theta_{guess})^2, \tag{10}$$

where ξ and θ_{guess} are two parameters. The first line of figure 3 shows the effects of those parameters on the L^2 regularisation of functional (4). The top-left picture shows the effect of ξ, the top-right shows the effect of θ_{guess}. Let us begin with figure 3 (top-left): it shows a regularised functional with ($\theta_{guess} = 0$ and) $\xi = 0.010, 0.029, 0.050$. As ξ increases, the slopes of the formerly zero gradient zones $]-\infty, -5]$ and $[10, \infty[$ are steeper and steeper. A GD algorithm will not remain stuck in those areas. On another hand, we also see that the regularised functional does not yield the same performances as the original one: in the vinicity of $\theta = 6$, the minimum increases with ξ. For $\xi = 0.05$, the local minimum in the vicinity of $\theta = 6$ is not anymore the best one and the regularised functional does not yield equivalent performances as the non-regularised one. Nothing changes around $\theta = 0 = \theta_{guess}$. Figure 3 (top-right) shows the regularised functional (10) for ($\xi = 0.01$ and) $\theta_{guess} = 0, 1, 6$. The performance, in term of minimum, of J remains unchanged in the vicinity of θ_{guess} whereas ξ ensures having non-zero gradient. But if θ_{guess} is not well enough chosen, the performances can greatly be altered: for $\theta_{guess} = 0, 1$, the minimum is higher, i.e. have poorer performances, than for the non-regularised functional.

Besides, it is not because the regularised functional now has some non-zero gradient regions that it is easier to minimise with a GD algorithm: the two last lines of figure 3 show how the GD algorithm behaves for two different values of $\xi = 0.010, 0.029$ (for $\theta_{guess} = 0$). On the second line, the gradient are so steep that (for the chosen learning rate) the algorithm diverges. For smoother gradients (last line of figure 3), the algorithm path remains bounded but still presents an unstable behaviour. Of course, if, despite those instabilities and this diverging behaviour, the GD algorithm comes close to the minimum, a good candidate may be found and kept in memory (cf. the blue dot • which is close to the minimum of the regularised functional). But surely, many iterations are lost. We insist on the fact that the oscillations here are numerical instabilities (the algorithm is deterministic, the oscillations are not due to the introduction of any stochasticity).

Example 3.1.6 (The regularisation effect, quantitative results) *The results of example 3.1.5 related to figure 3 were mainly qualitative. We here suggest some quantitative ones.*

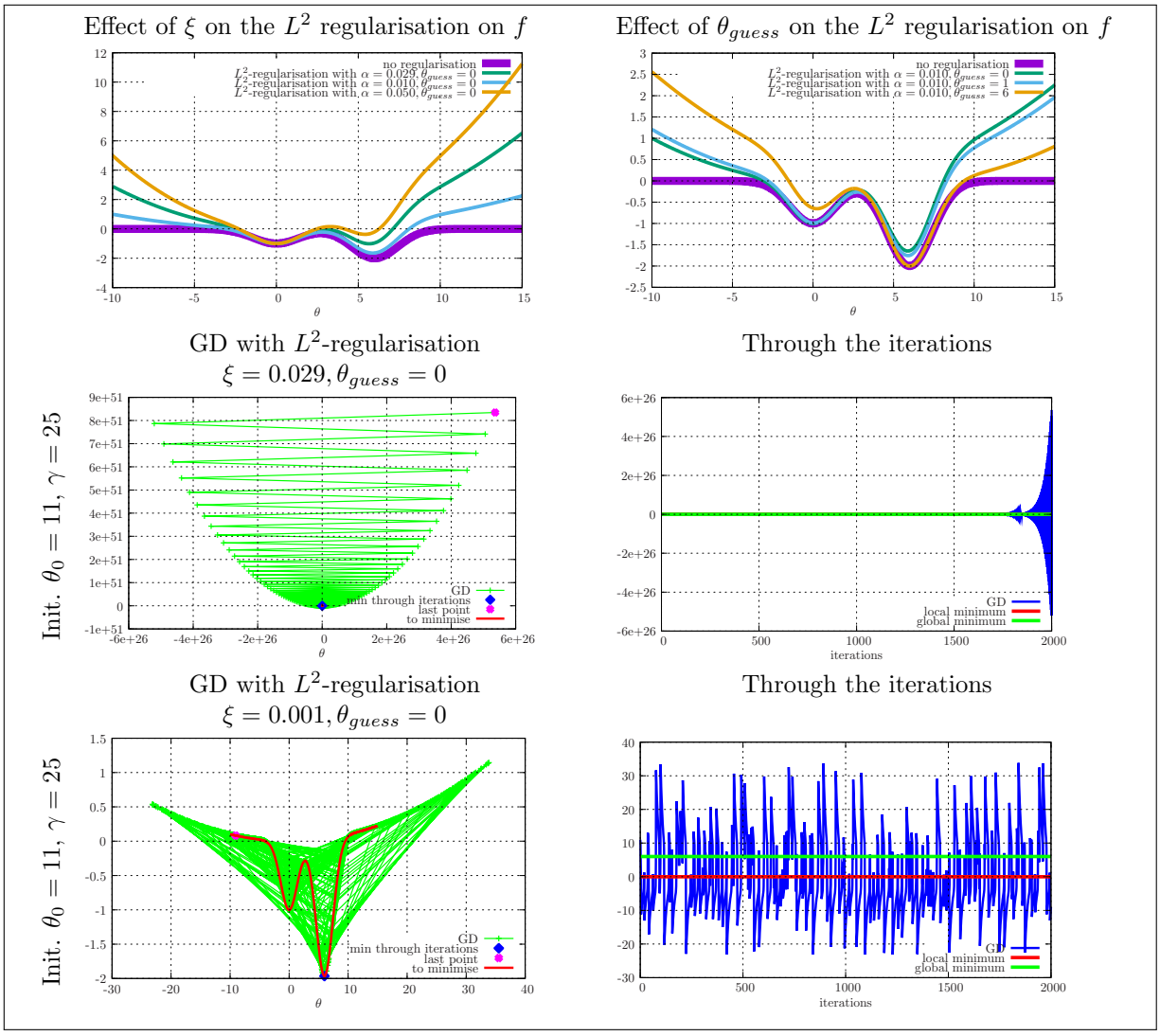

Figure 3: These figures revisit the test-case of example 3.1.1 but with a GD algorithm with learning rates $\gamma = 25.0$ with the minimisation of an L^2-regularised functional (see example 3.1.5).

The conditions are the same as in the previous (quantitative) examples 3.1.2–3.1.4. The probabilities for the penalised GD algorithm to reach local/global minima vicinities are displayed in table 3. The regularised results must be compared to the last line of table 2: regularisation does improve the results for $\gamma = 25.0$ but the probabilities to recover the vicinities remain quite low. The reasons are those depicted in figure 3 and detailed in example 3.1.5.

3.1.2 ... To stochastic optimisation algorithms

Let us now tackle the problem of the dimension of the hessian (which we avoided in example 3.1.1 which is 1D and for which we computed J'' analytically). The size of the hessian depends on the size $Card(\theta)$ of the vector of weights. This size depends on the number of neurons per layer and

Initialiser $\theta_0 \sim \mathcal{U}([-15, 15])$	Vicinity of $\theta^* = 0$	Vicinity of $\theta^* = 6$
penalised GD ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10^{-2}, 10^{-2}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-2}, 6 + 10^{-2}])$
$\gamma = 25.0$ $\xi = 0.029$	$0.56\% \in [0.413\%, 0.707\%]$	$0.40\% \in [0.276\%, 0.524\%]$
$\gamma = 25.0$ $\xi = 0.001$	$0.00\% \in [0.000\%, 0.000\%]$	$0.21\% \in [0.204\%, 0.221\%]$

Table 3: Probabilities for the penalised GD algorithm to recover the local ($\theta^* = 0$) or the global ($\theta^* = 6$) minimum of (4) for $\gamma = 25.0$ and two values of the regularisation coefficient ξ. To compute the probabilities, we have resort to $N_{seed} = 10^4$ multiple initialisations with $\theta_0 \sim \mathcal{U}([-15, 15])$. Confidence intervals for the results are provided. The 0.00% are not significative but attest for a small probability. Comments are provided in example 3.1.6.

on the number of layers. Even if growing polynomially with the dimension, the number of weights remains important mainly due to the important number of input dimensions (d_{in}) we aim at taking into account: for example, suppose we aim at working with datasets composed of 100×100 pixel images, then a neural network with only one layer with n_c neurons needs to tune $(d_{in} + 1) \times n_c + n_c \times d_{out} = 10001 \times n_c + 10000 \times d_{out}$ weights. Even for small number of neurons n_c and of outputs d_{out}, this number implies a diagonalisation/inversion of a huge matrix.

Now comes the second reflex as a numerician: *when you have a dimension problem, a numerician uses a discretisation method which is the less sensitive to the dimension possible, the Monte-Carlo (MC) one [39, 40, 60]*. How do you do that? You can discretise PDEs with an MC method so let us build a PDE. Equation (6) which describes one step of a Newton algorithm is equivalent to

$$\begin{aligned} 0 = & \int L'(X,\theta)\nabla_\theta u(X,\theta)\,\mathrm{d}\mathcal{P}_X \\ & + \int \left[L'(X,\theta)\nabla^2_{\theta,\theta} u(X,\theta) + L''(X,\theta)\nabla_\theta u(X,\theta)\nabla^T_\theta u(X,\theta)\right](\theta^* - \theta)\,\mathrm{d}\mathcal{P}_X, \end{aligned} \tag{11}$$

where $L''(x,y) = \partial^2_{y,y} L(x,y)$ and where we introduced new notations, $L'(X,\theta) = L'(u(X), u(X,\theta))$ and $L''(X,\theta) = L''(u(X), u(X,\theta))$, for the sake of conciseness. Remember that in the previous expression,

- θ^* is the unknown,
- the equation is still vectorial at this stage,
- the equation is averaged over $\mathrm{d}\mathcal{P}_X$.
- In practice, $\mathrm{d}\mathcal{P}_X$ is replaced by $\mathrm{d}\mathcal{P}^N_X$ but the next analysis are independent of the distribution of training points (i.e. discretised or not).

Now, we are going to make $u(X,\theta)$ the unknown of an instationary PDE. We will consequently temporarily have two unknowns, $u(t,X,\theta)$ and θ^*.

Also, we are going to *temporarily* forget about the integration with respect to $\mathrm{d}\mathcal{P}_X$. We will come back to it in section 3.4. Let us now rewrite (11) for an arbitrary component $i \in \{1, ..., Card(\theta)\}$ and an arbitrary output $k \in \{1, ..., d_{out}\}$:

$$\begin{aligned} 0 = & \ \partial_{\theta_i} u_k(X,\theta)\left[L'(X,\theta) + L''(X,\theta)\sum_{j=1}^{Card(\theta)} \partial_{\theta_j} u_k(X,\theta)(\theta^*_j - \theta_j)\right] \\ & + L'(X,\theta)\sum_{j=1}^{Card(\theta)} \partial^2_{\theta_i,\theta_j} u_k(X,\theta)(\theta^*_j - \theta_j). \end{aligned} \tag{12}$$

The above equation reminds of a drift-diffusion equation. In the next section, we recall few results on that type of PDE. We aim at clarifying some analogies between parabolic PDEs and (12).

3.2 Few reminders on the stochastic resolution of parabolic equations

In this section, we recall few results which will be handy later on when studying the optimisation algorithms at play in ML. Let us first recall the general Ito formula.

Lemma 1 (The general Ito formula) *This definition is taken from [53, 70]. Assume X_t is an n dimensional Ito process. Let $t, x \in \mathbb{R}^+ \times \mathbb{R}^n \longrightarrow g(t,x) = (g_1(t,x), ..., g_p(t,x))^T \in \mathbb{R}^p$ with g being $\mathcal{C}^2$. Then $Y_t = g(t, X_t)$ is a p dimensional Ito process and verifies $\forall k \in \{0, ..., p\}$*

$$\mathrm{d}Y_t^k = \partial_t g_k(t, X_t)\,\mathrm{d}t + \sum_{i=1}^n \partial_{x_i} g_k(t, X_t)\,\mathrm{d}X_t^i + \frac{1}{2}\sum_{i,j=1}^n \partial^2_{x_i,x_j} g_k(t, X_t)\,\mathrm{d}X_t^i\,\mathrm{d}X_t^j.$$

From the above general Ito formula can be deduced Ito's lemma recalled here.

Lemma 2 (Ito's lemma [53, 70]) *Define*

$$\mathrm{d}X_t = \mu(t, X_t)\,\mathrm{d}t + \sigma(t, X_t)\,\mathrm{d}B_t, \tag{13}$$

where $(t,x) \in \mathbb{R}^+ \times \mathbb{R}^n \longrightarrow \mu(t,x) \in \mathbb{R}^n$ and $(t,x) \in \mathbb{R}^+ \times \mathbb{R}^n \longrightarrow \sigma(t,x) \in \mathbb{R}^{n\times m}$ and $\mathrm{d}B_t$ is an m dimensional Brownian process, then we have $\forall k \in \{1, ..., p\}$

$$\begin{aligned}\mathrm{d}Y_t^k \quad &= \left[\partial_t g_k(t, X_t) + \sum_{i=1}^n \partial_{x_i} g_k(t, X_t)\mu_i(t, X_t) + \frac{1}{2}\sum_{i,j=1}^n \partial^2_{x_i,x_j} g_k(t, X_t)\left[\sigma(t, X_t)\sigma^T(t, X_t)\right]_{i,j}\right]\mathrm{d}t \\ &+ \sum_{i=1}^n \partial_{x_i} g_k(t, X_t)\sum_{j=1}^m \sigma_{i,j}(t, X_t)\,\mathrm{d}B_t^j,\end{aligned} \tag{14}$$

where $\left[\sigma(t, X_t)\sigma^T(t, X_t)\right]_{i,j} = \sum_{k=1}^m \sigma_{i,k}(t, X_t)\sigma_{j,k}(t, X_t)$. We insist on the fact that m and p are independent. In matricial form (more concise but the dimensions are less visible), we get

$$\begin{aligned}\mathrm{d}Y_t = \quad &\left[\partial_t g(t, X_t) + \nabla_x g(t, X_t)\mu(t, X_t) + \frac{1}{2}\mathrm{Tr}\left[\sigma^T(t, X_t)\nabla^2_{x,x} g(t, X_t)\sigma(t, X_t)\right]\right]\mathrm{d}t \\ &+\nabla_x g(t, X_t)\sigma(t, X_t)\,\mathrm{d}B_t.\end{aligned} \tag{15}$$

Similarly, we recall Kolmogorov's theorem in the next lines.

Theorem 1 (The Kolmogorov backward equation [53, 70]) *Let $u^0(x)$ be $\mathcal{C}^2$ and let X_t^x be an Ito process as in (13) with initial condition $X_0 = x$. Kolmogorov's theorem states that $u(t,x)$ defined as $u(t,x) = \mathbb{E}[u^0(X_t^x)]$ is solution of*

$$\begin{cases} \partial_t u(t,x) + \sum_{i=1}^n \mu_i(t,x)\partial_{x_i} u(t,x) + \frac{1}{2}\sum_{i,j=1}^n [\sigma(t,x)\sigma^T(t,x)]_{i,j}\partial^2_{x_i,x_j} u(t,x) = 0, \\ u(x,0) = u^0(x). \end{cases} \tag{16}$$

The above theorem states that we can solve a drift-diffusion equation by averaging realisations of a stochastic process. This will be of interest later on. Let us first continue with useful results.

Theorem 2 (The Kolmogorov forward equation [53, 70]) *Let X_t^x be an Ito process as in (13). Let us denote by $p_x(t,y)$ the distribution of process X_t^x (beginning at $X_0 = x$), i.e. such that*

$$\mathbb{E}[f(X_t^x)] = \int f(y)p_x(t,y)\,\mathrm{d}y,$$

for all measurable f. Then $p_x(t,y)$ satisfies the Kolmogorov forward (Fokker-Planck) equation

$$\begin{cases} \partial_t p_x(t,y) + \sum_{i=1}^n \partial_{y_i}(\mu_i(y)p_x(t,y)) - \frac{1}{2}\sum_{i,j=1}^n \partial^2_{y_i,y_j}([\sigma(t,y)\sigma^T(t,y)]_{i,j}p_x(t,y)) = 0, \\ p_x(0,y) = p_{0,x}(y). \end{cases} \tag{17}$$

Note that the linear operator in (17) is the adjoint of the one in (16). The Fokker-Planck equation gives an idea of the distribution at time t of the position of X_t when departing from x.

Let us finally insist on the fact that the results presented in this section all concern linear PDEs whereas (12) is nonlinear. The material of this section will only be relevant once a linearisation strategy for (12) chosen. We tackle this point and the analogy between ML algorithms and PDE resolutions in the next section.

3.3 From stationary to unstationary processes and their discretisations

Equation (12) is close to equation (16). We aim at transforming (12) into something which even more looks like (16). Let us insist on several points:

- first, in Ito's lemma, $g \in \mathbb{R}^p$ whereas $u \in \mathbb{R}^{d_{out}}$ so that p in Ito's lemma echoes the number of outputs of the ANN. Of course, n plays the role of $Card(\theta)$.
- Second, (12) is still vectorial: it lacks a sum if compared to (14). Let us introduce an arbitrary vector $\alpha \in \mathbb{R}^{Card(\theta)}$. Then (11) is equivalent to $\forall \alpha \in \mathbb{R}^{Card(\theta)}$,

$$\begin{aligned} 0 = & \int L'(X,\theta)\alpha^T \nabla_\theta u(X,\theta)\, \mathrm{d}\mathcal{P}_X \\ & + \int \alpha^T \left[L'(X,\theta)\nabla^2_{\theta,\theta} u(X,\theta) + L''(X,\theta)\nabla_\theta u(X,\theta)\nabla^T_\theta u(X,\theta) \right] (\theta^* - \theta)\, \mathrm{d}\mathcal{P}_X. \end{aligned} \tag{18}$$

Remark 3.1 *Equivalency between (11) and (18) is only ensured $\forall \alpha \in \mathbb{R}^{Card(\theta)}$ or for α the components of a basis of $\mathbb{R}^{Card(\theta)}$.*

- Third, there is one less and one additional dependence: θ in (12) echoes x in (16). But (12) does not depend on time t (yet) and (16) does not depend on X. The dependence with respect to X will be tackled in section 3.4 but assume that the solution of (16) depends on X, then it has exactly the same structure as some uncertainty quantification problems, see [60, 59, 57, 12, 55].

  Let us here focus on the dependence with respect to time t. Let us now consider an instationary equation (typically, we suppose that u now also depends on time t). Then, for one component of (18), we have $\forall k \in \{1, .., d_{out}\}$

$$\begin{aligned} 0 = \; & \partial_t u_k(t,X,\theta) + \sum_{i=1}^{Card(\theta)} \partial_{\theta_i} u_k(t,X,\theta) \left[\alpha_i L'(t,X,\theta) + L''(t,X,\theta)\alpha_i \sum_{j=1}^{Card(\theta)} \partial_{\theta_j} u_k(t,X,\theta)(\theta^*_j - \theta_j) \right] \\ & + \frac{1}{2} \sum_{i,j=1}^{Card(\theta)} \partial^2_{\theta_i,\theta_j} u_k(t,X,\theta) \left[2L'(t,X,\theta)\alpha_i(\theta^*_j - \theta_j) \right]. \end{aligned} \tag{19}$$

The above equation is

- now scalar,
- must be true $\forall \alpha \in \mathbb{R}^{Card(\theta)}$ or for a set of basis vectors of the same space (see remark 3.1),
- and must be solved for $\forall t \in [0,T]$ and $\forall \theta \in \Theta$. Furthermore, if
  1. $T < \infty$ and[9] $|\Theta| \leq \infty$, (19) together with an initial condition $u^0_k(\theta) = u_k(t=0,\theta), \forall k \in \{1, ..., d_{out}\}$ is a Cauchy problem.

[9] $|\Theta|$ denotes the volume of the space Θ.

2. $T = \infty$ and $|\Theta| < \infty$, (19) must be completed with non-zero boundary condition in order to have a non-zero solution as time tends to ∞.

In both cases, the problem is wellposed under the conditions detailed below. In ML problems, in general, the space of the parameters Θ is not bounded (i.e. $|\Theta| = \infty$) so we will mainly consider the first case above. Still, the following discussions are also valid in the second case.

– Besides, the resolution of the equation must be instrumented in order to monitor the generalisation error $\overline{J}$: due to the fact that u now also depends on time t, the generalisation error (1) also does and is now given by

$$\overline{J}(t,\theta) = \overline{J}(u(t,\cdot,\theta)) = \int L(u(X), u(t,X,\theta))\, \overline{\mathrm{d}\mathcal{P}_X^N}.$$

During the resolution, we must find the couple $(t^*, \theta^*) \in [0,T] \times \Theta$ such that $\overline{J}(t^*,\theta^*) \leq \overline{J}(t,\theta), \forall (t,\theta) \in [0,T] \times \Theta$.

Remark 1 *The above points highlight the fact that ML frameworks (such as TensorFlow or PyTorch or SciKitLearn etc.) are very similar to our simulation codes for computational physics. Let us give an example:*

– *at the CEA, we develop codes simulating solid mechanics. We solve the (hyperbolic) equations of continuum dynamics [44, 45, 33, 34]:*

$$\begin{cases} \partial_t \tau(t,X,\theta) = \nabla_\theta v(t,X,\theta), \\ \partial_t v(t,X,\theta) = \nabla_\theta \Sigma(t,X,\theta), \\ \partial_t e(t,X,\theta) = \nabla_\theta \left[\Sigma^T(t,X,\theta) v(t,X,\theta)\right], \end{cases}$$

where $\tau \in \mathbb{R}^+$ is the specific volume, $v \in \mathbb{R}^3$ is the velocity, $e \in \mathbb{R}^+$ is the total energy and the system is closed thanks to a constitutive law $\Sigma(\tau, e) \in \mathbb{R}^{3\times 3}$.

– *We typically look for the vector of unknowns $u = (\tau, v, e)^T$ for times $t \in [0,T] \subset \mathbb{R}^+$, for the spatial variable $\theta \in \Theta \subset \mathbb{R}^3$ and with $X \sim \mathrm{d}\mathcal{P}_X$ modeling some uncertainties on the initial/boundary conditions or the constitutive law (Σ) [35, 36, 37, 38, 66, 22, 67, 60, 21].*

– *Quantity $\Sigma(t,X,\theta)$ is the* constitutive law*: it is chosen in order to model the physical behaviour of a material. It is chosen as a function of τ, u, e, i.e. $\Sigma(t,X,\theta) = \Sigma(\tau(t,X,\theta), u(t,X,\theta), e(t,X,\theta))$, in order to close the system*[10]*.*

– *In computational physics, it is common monitoring a functional of the physical variables in order to perform a study. For example, in our solid mechanics context, $\overline{J}(t,\theta)$ would be the minimum over time $t \in [0,T]$ and space $\theta \in \Theta$ of the main stress (the minimum of the eigenvalues of Σ). Thanks to this physical quantity, we are able to know whether our material will break or not at some time t^* and position θ^* which are* a priori *unknown.*

In other words, ML frameworks are just classical codes, solving a particular (uncertain) PDE (a parabolic one) given by (19) together with relevant modeling assumptions (mainly on $\alpha, \theta^* - \theta$). Now, there are several ways to solve the PDE of interest: you can have resort to deterministic or stochastic (MC) schemes.

[10] i.e. in order to have the same number of equations and unknowns.

Let us complete the analogy with (14). The next question is: can we identify μ, σ as in (14) but in (19)? For μ, this is pretty obvious, see below,

$$\left\{\begin{array}{l} 0 = \partial_t u_k(t,X,\theta) \\ \quad + \sum_{i=1}^{Card(\theta)} \partial_{\theta_i} u_k(t,X,\theta) \underbrace{\left[\alpha_i L'(t,X,\theta) + L''(t,X,\theta)\alpha_i \sum_{j=1}^{Card(\theta)} \partial_{\theta_j} u_k(t,X,\theta)(\theta_j^* - \theta_j)\right]}_{\mu_i(t,X,\theta)} \\ \quad + \frac{1}{2} \sum_{i,j=1}^{Card(\theta)} \partial^2_{\theta_i,\theta_j} u_k(t,X,\theta) \underbrace{\left[2L'(t,X,\theta)\alpha_i(\theta_j^* - \theta_j)\right]}_{[\sigma(t,X,\theta^*-\theta)\sigma^T(t,X,\theta^*-\theta)]_{i,j}\ ??} , t \in [0,T], \theta \in \Theta, X \sim \mathrm{d}\mathcal{P}_X, \\ u_k(t=0,X,\theta) = u_k^0(X,\theta), \forall k \in \{1,...,d_{out}\}, \theta \in \Theta, X \sim \mathrm{d}\mathcal{P}_X, \\ \text{together with the evaluation of } \overline{J}(t,\theta) = \overline{J}(u_1(t,\cdot,\theta),...,u_{d_{out}}(t,\cdot,\theta)) \text{ in order to find} \\ t^* \in [0,T], \theta^* \in \Theta \text{ such that } \overline{J}(t^*,\theta^*) \leq \overline{J}(t,\theta), \forall t \in [0,T], \forall \theta \in \Theta. \end{array}\right. \tag{20}$$

But it is not clear whether (see the ?? in the above expression) the coefficients of the second order operator can be rewritten as the general term of a matrix times its transpose. In other words, it is not guaranted that all choices of $\alpha, \theta^* - \theta, L$ ensure wellposedness of the PDE we need to solve.

Remark 3.2 *The existence of the above decomposition is not straightforward: we must have*

$$\left[\sigma\sigma^T\right]_{i,j} = \sum_{k=1}^{m} \sigma_{i,k}\sigma_{j,k} = 2L'(t,X,\theta)\alpha_i(\theta_j^* - \theta_j). \tag{21}$$

- *If $Card(\theta) = 1$ then it exists under positivity conditions on L', α and $\theta^* - \theta$.*
- *In dimension 2, it is easy building a set of parameters such that the decomposition does not exist.*
- *For a general dimension $Card(\theta)$, it is always possible to*
  - *choose $\alpha = (\theta^* - \theta)$,*
  - *assume positiveness of L' and $\theta^* - \theta$ component by component,*
  - *and build $\sigma(t,X,\theta^* - \theta) = diag(\sqrt{2L'(t,X,\theta)}|\theta_1^* - \theta_1|, ..., \sqrt{2L'(t,X,\theta)}|\theta_m^* - \theta_m)|)$.*
  - *With the above conditions, (20) has the desired structure. Note that in this case, many terms are zero in $[\sigma\sigma^T]_{i,j}$ and m is arbitrary (it can be smaller than $Card(\theta)$ or bigger which would imply having many zero on the diagonal).*
- *Of course, uniqueness of the existing decomposition is not ensured: one can choose $m \in \mathbb{N}$ in the previous diagonal decomposition.* **Every modeling[11] choices of $\alpha, \theta^* - \theta$ make a potential new heuristic in order to compute our minimum.** *Of course, just as in computational physics, not every modeling choices are equivalent: some are coarse but efficient and enough in some situations, some are fine and computationally intensive but necessary in order to obtain accurate results. This will be emphasized in section 3.5.*

The positiveness or the use of absolute values for L' or the components of $\theta^* - \theta$ echoe the discussion on the sign of the eigenvalues of the hessian in the previous section[12]: by construction, the $\sigma\sigma^T$ symmetrical structure will make sure we look for minima (and not maxima).

[11] Echoing the physical modeling choice operated *via* Σ in remark 1.

[12] and taking the absolute value $|H|$ of the hessian H in a Newton algorithm in order to go in the opposite direction of the gradient direction.

In the following sections, we assume the existence of $\sigma(t, X, \theta^* - \theta, \alpha)$ as desired, even if some choices and hypothesis need to be made. As in practice, we can always choose the decomposition as in the third point of remark 3.2 and from now on, we will assume wellposedness for the ML problem/PDE (20). We can consequently try to solve it.

At this stage, the reader may wonder why all this fuss, all these hypothesis stated in the more general terms possible etc. Mainly because numericians are more familiar with selecting a set of PDEs, analysing its properties *before* choosing a discretisation method. In other words, only now is the time for the discretisation question: due to the structure of the problem of interest, we are able to relate Ito's lemma to (20). Regarding (20) and lemma 1, we are tempted to identify

- the drift term as

$$\mu(t, X, \theta, \alpha) = \alpha L'(t, X, \theta) + L''(t, X, \theta)\alpha\nabla_\theta^T u(t, X, \theta)(\theta^* - \theta),$$

- and its diffusion coefficients $\left[\sigma(t, \alpha, X, \theta^* - \theta)\sigma^T(t, \alpha, X, \theta^* - \theta)\right]_{i,j} = 2L'(t, X, \theta)\alpha_i(\theta_j^* - \theta_j)$.

- In order to solve equation (20) (but still, not exactly our optimisation problem! We still do not integrate with respect to X!), we can consequently introduce the stochastic process

$$\mathrm{d}\theta_t(X) = \left[\alpha L'(t, X, \theta_t) + L''(t, X, \theta_t)\alpha\nabla_\theta^T u(t, X, \theta_t)(\theta^* - \theta_t)\right]\,\mathrm{d}t + \sigma(t, X, \alpha, \theta^* - \theta_t)\,\mathrm{d}B_t. \quad (22)$$

  The instrumentation $\overline{J}(t, \theta)$ then becomes[13] $\overline{J}(t, \theta) = \overline{J}(\theta_t)$.
  The above equation (22) is a stochastic nonlinear ODE. It can not, in general, be integrated analytically. We must have resort to a numerical scheme. Let us consider the simplest integration scheme for an ODE, the explicit Euler scheme. It supposes integrating (22) over a time step $[0, t]$

$$\begin{aligned}\theta_t(X) = \theta_0(X) \quad &+ \int_0^t \left[\alpha L'(s, X, \theta_s(X)) + L''(s, X, \theta_s(X))\alpha\nabla_\theta^T u(s, X, \theta_s(X))(\theta^* - \theta_s(X))\right]\,\mathrm{d}s \\ &+ \int_0^t \sigma(s, X, \alpha, \theta^* - \theta_s(X))\,\mathrm{d}B_s,\end{aligned}$$

  and assuming

$$\begin{aligned}\alpha L'(s, X, \theta_s(X)) + L''(s, X, \theta_s(X))\alpha\nabla_\theta^T u(s, X, \theta_s(X))(\theta^* - \theta_s(X)) \approx \\ \alpha L'(0, X, \theta_0(X)) + L''(0, X, \theta_0(X))\alpha\nabla_\theta^T u(0, X, \theta_0(X))(\theta^* - \theta_0(X)),\end{aligned}$$

  $\forall s \in [0, t = \Delta t]$ together with

$$\sigma(s, X, \alpha, \theta^* - \theta_s(X)) \approx \sigma(0, X, \alpha, \theta^* - \theta_0(X)),$$

  $\forall s \in [0, t = \Delta t]$. As a consequence, for an arbitrary time interval $[t^n, t^n + \Delta t = t^{n+1}]$, we have

$$\begin{aligned}\theta_{n+1}(X) = \theta_n(X) \quad &+\Delta t\left[\alpha L'(t^n, X, \theta_n(X)) + L''(t^n, X, \theta_n(X))\alpha\nabla_\theta^T u(t^n, X, \theta_n(X))(\theta^* - \theta_n(X))\right] \\ &+\sigma(t^n, X, \alpha, \theta^* - \theta_n(X))\sqrt{\Delta t}\mathcal{G}(0, 1).\end{aligned} \quad (23)$$

  For convergence and stability, the explicit Euler scheme demands a restriction on the time step [41, 11]. Theorems in [41, 11] states that if Δt is small enough, we have a converging approximation of Y_t toward X_t with convergence rate given by $\mathcal{O}(\sqrt{\Delta t})$. From the above definitions, we realize that the stability of (23) in order to discretise (22) is not straightforward and is closely related to restrictions on L, α, θ^*, X.

  We talked about the explicit Euler scheme demanding a restriction on the time step but of course, it is also possible to design many other different integration schemes (implicit Euler,

[13] Note that the dependence with respect to X complicates the analysis. But this will be clarified later on, mainly in section 3.4.

semi-implicit, high-order Runge-Kutta etc.) depending on the resolution needs (unconditional stability, stiffness of not etc.), now that we identified the PDE we want to solve.
In a discretised form, the instrumentation $\overline{J}(t,\theta)$ becomes $\overline{J}(t,\theta) = \overline{J}(\theta_t) \approx \overline{J}(\theta_{t^n}) \approx \overline{J}(\theta_n)$ (this will be clarified later on).

We here put forward one important advantage of having resort to such PDE resolution framework: several numerical schemes, bearing various properties, are at hand to solve (20) and may benefit ML optimisers. This point will be investigated later on (and even illustrated in example 3.6.1).

Let us finish this section by few remarks:

– From the Kolmogorov backward theorem 1, we can tell that the analytical integration of (22), given by

$$\begin{aligned}\theta_t(X) = \quad & +\theta_0(X) \\ & + \int_0^t \left[\alpha L'(s,X,\theta_s(X)) + L''(s,X,\theta_s(X))\alpha\nabla_\theta^T u(s,X,\theta_s(X))(\theta^* - \theta_s(X))\right] \mathrm{d}s \\ & + \int_0^t \sigma(s,X,\alpha,\theta^* - \theta_s(X))\,\mathrm{d}B_s,\end{aligned} \tag{24}$$

with time $t \in [0,T]$ makes sure that (18) is fulfilled for the chosen $\alpha \in \mathbb{R}^{Card(\theta)}$. But nothing ensures that *only one choice* of $\alpha \in \mathbb{R}^{Card(\theta)}$ will ensure that (11) is fulfilled, see remark 3.1.

– Note that, as written above, θ_t (hence θ_{t^*} such that $\overline{J}(\theta_{t^*}) \leq \overline{J}(\theta_t)\ \forall t \in [0,T]$) depends on X (i.e. $\theta_{t^*} \equiv \theta_{t^*}(X)$) whereas in practice in ML problems, θ_n does not depends on X (except maybe in [24] which will be briefly discussed in section 3.4). In ML problems, equation (20) is averaged with respect to $\mathrm{d}\mathcal{P}_X$. This is an important point and we did not avoid it. It is long to debate on so we suggest postponing this discussion to section 3.4 (note that it deserves a whole section).

– In the second line of (14), the last term $\nabla_x u(t,X_t)\sigma(t,X_t)\,\mathrm{d}B_t$ corresponds to an error term. With the analytical integration (24) of (22), theorem 1 tells that $\int_0^t \nabla_x u(t,\theta_t)\sigma(t,\theta_t)\,\mathrm{d}B_t = 0$ (martingale property). But if θ_t is discretised, nothing ensures this same term is zero and it gives an idea of how the error may evolve with the time step $[0, t = \Delta t]$.

– In the Kolmogorov forward equation, μ is also called a *background*. From the Kolmogorov forward equation, one can tell that asymptotically as $t \to \infty$ (and given some boundary conditions on Θ), the distribution $p_{\theta_0}(t,\theta) \underset{t\to\infty}{\sim} p^\infty_{\theta_0}(\theta)$ of X_t satisfies[14]:

$$-\nabla_\theta(\mu^\infty(X,\alpha,\theta)p^\infty_{\theta_0}(X,\theta)) + \frac{1}{2}\nabla^2_{\theta,\theta}(\sigma^\infty(\alpha,X,\theta^* - \theta)((\sigma^\infty)^T(\alpha,X,\theta^* - \theta)p^\infty_{\theta_0}(X,\theta)) = 0.$$

The convergence of $p_{\theta_0}(t,\theta)$ toward $p^\infty_{\theta_0}(\theta)$ is exponential and the constant within the exponential is the same as in the Poincaré inequality related to μ, σ, see [56]. At this stage, theorem 1 and 2 of [73] may become handy (expressed in a UQ context, see section 3.4), in order to understand the behaviour with respect to time of $p_{\theta_0}(t,\theta)$ or its asymptotic behaviour $p^\infty_{\theta_0}(\theta)$ (but this is beyond the scope of this document).

From now on, we only considered what is under the integration $\int \cdot\, \mathrm{d}\mathcal{P}_X$ in (18). In the next section, we focus on this important point, the integration with respect to $\mathrm{d}\mathcal{P}_X$.

[14] boundary conditions should be given for wellposedness.

3.4 The dependence with respect to X in a PDE framework

Until now, we mainly focused on equation (20). For the sake of conciseness and without loss of generality, let us assume that there is only one output so that $d_{out} = 1$ and we do not need the k index. Then, $\forall X \sim \mathrm{d}\mathcal{P}_X$, (20) is rewritten as

$$\left\{\begin{array}{l} 0 = \partial_t u(t,X,\theta) \\ \quad + \displaystyle\sum_{i=1}^{Card(\theta)} \partial_{\theta_i} u(t,X,\theta) \left[\alpha_i L'(t,X,\theta) + L''(t,X,\theta)\alpha_i \sum_{j=1}^{Card(\theta)} \partial_{\theta_j} u(t,X,\theta)(\theta_j^* - \theta_j) \right] \\ \quad + \dfrac{1}{2} \displaystyle\sum_{i,j=1}^{Card(\theta)} \partial^2_{\theta_i,\theta_j} u(t,X,\theta) \left[2L'(t,X,\theta)\alpha_i(\theta_j^* - \theta_j) \right], t \in [0,T], \theta \in \Theta, X \sim \mathrm{d}\mathcal{P}_X, \\ u(t=0,X,\theta) = u^0(X,\theta), \theta \in \Theta, X \sim \mathrm{d}\mathcal{P}_X, \\ \text{together with the evaluation of } \overline{J}(t,\theta) = \overline{J}(u(t,\cdot,\theta)) \text{ in order to find} \\ t^* \in [0,T], \theta^* \in \Theta \text{ such that } \overline{J}(t^*,\theta^*) \leq \overline{J}(t,\theta), \forall t \in [0,T], \forall \theta \in \Theta. \end{array}\right. \tag{25}$$

At this stage, it is important putting forward one last time the analogy between the structure of (25) and the structure of Stochastic PDEs for uncertainty quantification, see [60, 59, 57, 12, 55]. The numerical methods described in the previous paper could certainly benefit the resolution of our ML problem. But in this document, we mainly would like to formalise what is commonly done in ML frameworks: the main problem with (25) and the analysis of the previous sections is that, in ML problems, we do not have to cancel[15] (25) $\forall X \sim \mathrm{d}\mathcal{P}_X$ but only (25) averaged over $\mathrm{d}\mathcal{P}_X$, i.e., only

$$\begin{aligned} 0 = \quad & \partial_t \int u(t,X,\theta)\, \mathrm{d}\mathcal{P}_X \\ & + \sum_{i=1}^{Card(\theta)} \int \partial_{\theta_i} u(t,X,\theta) \left[\alpha_i L'(t,X,\theta) + L''(t,X,\theta)\alpha_i \sum_{j=1}^{Card(\theta)} \partial_{\theta_j} u(t,X,\theta)(\theta_j^* - \theta_j) \right] \mathrm{d}\mathcal{P}_X \\ & + \frac{1}{2} \sum_{i,j=1}^{Card(\theta)} \int \partial^2_{\theta_i,\theta_j} u(t,X,\theta) \left[2L'(t,X,\theta)\alpha_i(\theta_j^* - \theta_j) \right] \mathrm{d}\mathcal{P}_X. \end{aligned} \tag{26}$$

Let us decompose $u(t,X,\theta)$ into

$$u(t,X,\theta) = \overline{u}(t,\theta) + \hat{u}(t,X,\theta),$$

where

$$\overline{u}(t,\theta) = \int u(t,X,\theta)\, \mathrm{d}\mathcal{P}_X \text{ and } \hat{u}(t,X,\theta) = u(t,X,\theta) - \overline{u}(t,\theta).$$

In fact, $\overline{u}$ is nothing more than the mean of u over $\mathrm{d}\mathcal{P}_X$ and $\hat{u}$ is a centered fluctuation. This kind of decomposition is intensively used in turbulence modeling, see [46]. We then have to solve

[15] Of course, if it is cancelled $\forall X \sim \mathrm{d}\mathcal{P}_X$ then it implies it will cancel the mean but this is not reciprocal, except with some more of less strong assumptions.

$$
\left\{
\begin{array}{l}
0 = \partial_t \overline{u}(t,\theta) \\
+ \displaystyle\sum_{i=1}^{Card(\theta)} \partial_{\theta_i}\overline{u}(t,\theta) \int \left[\begin{array}{l} \alpha_i L'(t,X,\theta) \\ +L''(t,X,\theta)\alpha_i \displaystyle\sum_{j=1}^{Card(\theta)} \partial_{\theta_j}\left[\overline{u}(t,\theta)+\hat{u}(t,X,\theta)\right](\theta_j^*-\theta_j) \end{array} \right] \mathrm{d}\mathcal{P}_X \\
+\frac{1}{2}\displaystyle\sum_{i,j=1}^{Card(\theta)} \partial^2_{\theta_i,\theta_j}\overline{u}(t,\theta) \int \left[2L'(t,X,\theta)\alpha_i(\theta_j^*-\theta_j)\right] \mathrm{d}\mathcal{P}_X \\
+ \displaystyle\sum_{i=1}^{Card(\theta)} \int \partial_{\theta_i}\hat{u}(t,X,\theta) \left[\begin{array}{l} \alpha_i L'(t,X,\theta) \\ +L''(t,X,\theta)\alpha_i \displaystyle\sum_{j=1}^{Card(\theta)} \partial_{\theta_j}\left[\overline{u}(t,\theta)+\hat{u}(t,X,\theta)\right](\theta_j^*-\theta_j) \end{array} \right] \mathrm{d}\mathcal{P}_X \\
+\frac{1}{2}\displaystyle\sum_{i,j=1}^{Card(\theta)} \int \partial^2_{\theta_i,\theta_j}\hat{u}(t,X,\theta)\left[2L'(t,X,\theta)\alpha_i(\theta_j^*-\theta_j)\right] \mathrm{d}\mathcal{P}_X, t\in[0,T], \theta\in\Theta, \\
\overline{u}(t=0,\theta)+\hat{u}(t=0,X,\theta) = \overline{u}^0(\theta)+\hat{u}^0(\theta,X), \theta\in\Theta, X\sim \mathrm{d}\mathcal{P}_X \\
\text{together with the evaluation of } \overline{J}(t,\theta) = \overline{J}(\overline{u}(t,\theta)+\hat{u}(t,\cdot,\theta)) \text{ in order to find} \\
t^*\in[0,T], \theta^*\in\Theta \text{ such that } \overline{J}(t^*,\theta^*)\leq \overline{J}(t,\theta), \forall t\in[0,T], \forall\theta\in\Theta.
\end{array}
\right. \tag{27}
$$

The above PDE is not closed in the sense that we have one equation but two unknowns $\overline{u}$ and $\hat{u}$. The same problem occurs in turbulence modeling, see [46]. In other words, we are going to need at least one additional modeling hypothesis/heuristic in order to close our system (as we have 2 unknowns but still only one equation).

Remark 3.3 *From the Kolmogorov backward theorem 16, we can tell that averaging the stochastic process*

$$
\begin{array}{rl}
\overline{\theta}_t = \overline{\theta}_0 & + \displaystyle\iint_0^t \left[\alpha L'(s,X,\overline{\theta}_s) + L''(s,X,\overline{\theta}_s)\alpha\nabla_\theta^T u(s,X,\overline{\theta}_s)(\theta^*-\overline{\theta}_s)\right] \mathrm{d}\mathcal{P}_X \,\mathrm{d}s \\
& + \displaystyle\iint_0^t \sigma(s,X,\alpha,\theta^*-\overline{\theta}_s)\,\mathrm{d}\mathcal{P}_X\,\mathrm{d}B_s,
\end{array} \tag{28}
$$

will make sure that the three first lines (and only the first lines) of (27), i.e.

$$
\left\{
\begin{array}{l}
0 = \partial_t \overline{u}(t,\theta) \\
+ \displaystyle\sum_{i=1}^{Card(\theta)} \partial_{\theta_i}\overline{u}(t,\theta) \int \left[\alpha_i L'(t,X,\theta) + L''(t,X,\theta)\alpha_i \sum_{j=1}^{Card(\theta)} \partial_{\theta_j} u(t,X,\theta)(\theta_j^*-\theta_j)\right] \mathrm{d}\mathcal{P}_X \\
+\frac{1}{2}\displaystyle\sum_{i,j=1}^{Card(\theta)} \partial^2_{\theta_i,\theta_j}\overline{u}(t,\theta) \int \left[2L'(t,X,\theta)\alpha_i(\theta_j^*-\theta_j)\right] \mathrm{d}\mathcal{P}_X, t\in[0,T], \theta\in\Theta, \\
\overline{u}(t=0,\theta) = \overline{u}^0(\theta), \theta\in\Theta, \\
\text{together with the evaluation of } \overline{J}(t,\theta) = \overline{J}(\overline{u}(t,\theta)) \text{ in order to find} \\
t^*\in[0,T], \theta^*\in\Theta \text{ such that } \overline{J}(t^*,\theta^*)\leq \overline{J}(t,\theta), \forall t\in[0,T], \forall\theta\in\Theta,
\end{array}
\right. \tag{29}
$$

are cancelled for the chosen $\alpha\in\mathbb{R}^{Card(\theta)}$. In other words, we here implicitly made the modeling hypothesis $\hat{u}\equiv 0$. We then have $\overline{u}(t,\theta) = \mathbb{E}[\overline{u}^0(\overline{\theta}_t^\theta)]$. Discretised with an explicit Euler scheme over time step $[t^n, t^n+\Delta t = t^{n+1}]$, (28) becomes

$$
\begin{array}{rl}
\overline{\theta}_{n+1} = \overline{\theta}_n & + \Delta t \displaystyle\int \left[\alpha L'(t^n,X,\overline{\theta}_n) + L''(t^n,X,\overline{\theta}_n)\alpha\nabla_\theta^T u(t^n,X,\overline{\theta}_n)(\theta^*-\overline{\theta}_n)\right] \mathrm{d}\mathcal{P}_X \\
& + \sqrt{\Delta t}\left[\displaystyle\int \sigma(t^n,X,\alpha,\theta^*-\overline{\theta}_n)\,\mathrm{d}\mathcal{P}_X\right]\mathcal{G},
\end{array} \tag{30}
$$

where $\mathcal{G} \sim (\mathcal{G}_1(0,1),...,\mathcal{G}_m(0,1))^T$. *Of course, in practice,* $\mathrm{d}\mathcal{P}_X$ *is a discrete measure* $\mathrm{d}\mathcal{P}_X^N = \sum_{i=1}^N w_i \delta_{X_i}(X)$. *Assume furthermore that* $w_i = \frac{1}{N}, \forall i \in \{1,...,N\}$, *then* (30) *is equivalent to*

$$\begin{aligned}\overline{\theta}_{n+1} = \overline{\theta}_n \quad &+\frac{\Delta t}{N}\sum_{i=1}^N \left[\alpha L'(t^n, X_i, \overline{\theta}_n) + L''(t^n, X_i, \overline{\theta}_n)\alpha\nabla_\theta^T u(t^n, X_i, \overline{\theta}_n)(\theta^* - \overline{\theta}_n)\right] \\ &+\frac{\sqrt{\Delta t}}{N}\left[\sum_{i=1}^N \sigma(t^n, X_i, \alpha, \theta^* - \overline{\theta}_n)\right]\mathcal{G}.\end{aligned} \tag{31}$$

The above expression needs the evaluation of $\nabla_\theta^T u(t^n, X_i, \overline{\theta}_n)$ *which is not known. Let us assume that* $\nabla_\theta^T u(t^n, X_i, \overline{\theta}_n) \approx \nabla_\theta^T u_{ANN}(t^n, X_i, \overline{\theta}_n)$ *where* u_{ANN} *expresses the fact that* u *is replaced by an ANN architecture: we just introduced the need for* back propagation *and the evaluation of the gradient of* u*, in order to estimate some term in the drift (see [28]). Finally,* (31) *resumes to*

$$\begin{aligned}\overline{\theta}_{n+1} = \overline{\theta}_n \quad &+\frac{\Delta t}{N}\sum_{i=1}^N \left[\alpha L'(X_i, \overline{\theta}_n) + L''(X_i, \overline{\theta}_n)\alpha\nabla_\theta^T u_{ANN}(t^n, X_i, \overline{\theta}_n)(\theta^* - \overline{\theta}_n)\right] \\ &+\frac{\sqrt{\Delta t}}{N}\left[\sum_{i=1}^N \sigma(X_i, \alpha, \theta^* - \overline{\theta}_n)\right]\mathcal{G}.\end{aligned} \tag{32}$$

Back propagation can then be understood as a type of linearisation of equation (31) in order to obtain (32).

Now, nothing prevents us from simulating stochastic process (32) thanks to an MC resolution, every terms can be computed provided *some choices of* L, α *and of an additional equation for* $\theta^* - \theta$ *agreeing that we already implicitly chose* $\hat{u} \equiv 0$: those points will be tackled in the next section 3.5.
Regarding our instrumentation, it then becomes the *on-the-fly* numerical estimation of $\overline{J}(t,\theta) = \overline{J}(\overline{\theta}_t) \approx \overline{J}(\overline{\theta}_{t^n}) \approx \overline{J}(\overline{\theta}_n)$. The weights $\overline{\theta}_{n^*}$ such that $\overline{J}(\overline{\theta}_{n^*}) \leq \overline{J}(\overline{\theta}_n)\ \forall n \in \{1,...,n_{\text{epoch}}\}$ must be kept in memory.

Remark 3.4 *Note that if we attempt to cancel only the three first lines of* (27)*, i.e. the equation satisfied by* $\overline{u}$ *given by* (29)*, we only solve the minimisation problem if the fluctuations* $\hat{u}$ *of* u *are small, i.e. if* $u(t,X,\theta) \approx \overline{u}(t,\theta)$. *An attempt to characterise more precisely the situation when this property occurs will be made in section 4.*

With the few above pages, we have a nice framework to solve our optimisation problem. Let us compare it to what is commonly done in ML algorithms such as those available in TensorFlow, PyTorch, SciKitLearn etc.

3.5 Back to ML algorithms and analogies with PDE resolutions

Let us go back to our ML framework and study the algorithms available in the most popular softwares. Care will be taken to revisit them in the PDE framework discussed in the previous pages. Note that, usually, the description of the optimisation algorithms in ML always begins with the description of one iteration of a GD algorithm. Suppose there are N points in the training set $(X_i, w_i = \frac{1}{N})_{i\in\{1,...,N\}}$, then

$$\theta_{n+1} = \theta_n - \frac{\gamma}{N}\sum_{k=1}^N L'(X_k, \theta_n)\nabla_\theta u(X_k, \theta_n), \tag{33}$$

where γ is the learning rate, see [28].

Remark 3.5 *If we choose*

– $\alpha = -\nabla_\theta u(X,\theta)$,

– *and eliminate unknown* θ^* *by considering an additional equation* $\theta^* - \theta = 0$, $\forall \theta \in \mathbb{R}^{Card(\theta)}$,

– *then for arbitrary loss function* L,

we recover the GD algorithm and the learning rate γ *is nothing more than a time step* Δt *(this fact is already commonly accepted, see [18, 31]). Note that according to the PDE framework of section 3.2, averaging over (33) ensures solving (and instrumenting the resolution by computing* $\overline{J}(t,\theta)$*)*

$$\begin{aligned} 0 = \partial_t \overline{u}(t,\theta) - \sum_{i=1}^{Card(\theta)} \partial_{\theta_i}\overline{u}(t,\theta) \int \partial_{\theta_i} u(t,X,\theta) L'(t,X,\theta)\, \mathrm{d}\mathcal{P}_X, \\ \text{with } \overline{u}(t,\theta) = \int u(t,X,\theta)\, \mathrm{d}\mathcal{P}_X, \end{aligned} \tag{34}$$

with an MC scheme. The above PDE remains quite far from what we were aiming at solving with (29) for example. In a PDE framework, this is commonly called a lack of consistency of the numerical method. *But it does not mean the results with SGD are worthless. In many situations, GD does give accurate enough results. This can be explained by the fact that (34) may imply, under several suited conditions depending on* $\alpha, L, \theta^* - \theta, \hat{u}, \mathrm{d}\mathcal{P}_X$*, cancelling the gradient: under some conditions, the solution of (27) may coincide with the solution of (34). Assume for example that* $\alpha = -\nabla_\theta u$ *in (27), and that*

1. $L'' \sim \delta \sim 0$,
2. $\hat{u} \sim \delta \sim 0$,
3. *and* $L'\nabla_\theta u(\theta^* - \theta) \sim \delta \sim 0$ *with* $\theta^* - \theta$ *not necessarily zero,*

then (27) degenerates[16] *toward (34). In this case (the next points are different ways to understand the analysis we just made)*

– *solving (34) is equivalent to solving (27) in the regime* $\delta \sim 0$,

– *the solutions of (34) and of (27) coincides in the regime* $\delta \sim 0$,

– *if* $\delta \sim 0$*, it is enough solving (34) in order to recover the solution of (27).*

We will come back more in details to such analysis, which is classical in computational physics (cf. [68, 20, 1, 63] for examples) in section 4. Point 3 is interesting in the sense that it echoes the 'I am feeling lucky' error discussion of section 1: we can be far from the optimum, i.e. $\theta^* - \theta \sim \frac{1}{\delta} \gg 1$*, but if* L' *and the gradient of* u *are small, i.e. if* $L' \sim \delta$ *and* $\nabla_\theta u \sim \delta$*, then point 3 is still fulfilled as* $L' \times \nabla_\theta u \times (\theta^* - \theta) \sim \delta \times \delta \times \frac{1}{\delta} \sim \delta \sim 0$ *remains small.*
Note that the reinterpretation of GD in a PDE framework implies a particular choice of α*. Whether this choice ensures equivalency with the ML problem we aim at solving (see the discussion in remark 3.1) is far from being obvious.*

What may struck the numerician in this kind of presentation is that ML books almost (I may not be enough familiar with the extremely (!!) furnished litterature) always begin by a *discretised form of an uncharacterised PDE*. So, as briefly emphasized in remark 3.5, we are going to systematically perform some kind of *reverse engineering*. Note that the behaviour of (33) has been briefly studied in example 3.1.3 for $N = 1$ and particular $\alpha, \theta^* - \theta, \hat{u}, L$ implicitly defined.

[16] In the sense of a Hilbert/Chapman-Enskog development [29, 15] but this will be more rigorously detailed later on, cf. section 4.

In general, in classical ML books, just after the description of GD comes the description of Stochastic GD (SGD). It is justified by its efficiency. It can be described as follows: introduce a parameter $n \in \{1, ..., N\}$, which is called the *batch size*, then, an iteration of SGD beginning at iteration/epoch j is given by

Set $\theta_j^0 = \theta_j$
For l in $\{1, ..., \frac{N}{n} = K\}$ (one iteration of this loop is called a batch)
sample $(X_k^l, w_k^l)_{k\in\{1,...,n\}}$ points amongst the $(X_i, w_i)_{i\in\{1,...,N\}}$ according to measure $\mathrm{d}\mathcal{P}_X^{N,n}$,
$\theta_{j+1}^{l+1} = \theta_j^l - \gamma \sum_{k=1}^{n} w_k^l L'(X_k^l, \theta_j^l) \nabla_\theta u(X_k^l, \theta_j^l)$,
Then $\theta_{j+1} = \theta_{j+1}^K$.

Once again, what may appear strange to a numerician is that (33) is already a discretisation of

$$\theta_{j+1} = \theta_j - \gamma \int L'(X, \theta_j) \nabla_\theta u(X, \theta_j) \, \mathrm{d}\mathcal{P}_X = \theta_j - \gamma \sum_{k=1}^{N} w_k L'(X_k, \theta_j) \nabla_\theta u(X_k, \theta_j) + \mathcal{O}(N^\beta), \tag{35}$$

where $\mathcal{O}(N^\beta)$ is the asymptotic integration error[17] of the quadrature $(X_i, w_i)_{i\in\{1,...,N\}}$. And now, with SGD, we kind of introduced an approximation of the approximation as

$$\begin{aligned} \theta_{j+1}^{l+1} &= \theta_j - \gamma \sum_{k=1}^{N} w_k L'(X_k, \theta_j^l) \nabla_\theta u(X_k, \theta_j^l), \\ &= \theta_j^l - \gamma \left[\sum_{k=1}^{n} w_k^l L'(X_k^l, \theta_j^l) \nabla_\theta u(X_k^l, \theta_j^l) + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right) \right], \end{aligned} \tag{36}$$

which holds under mild assumptions on $\mathrm{d}\mathcal{P}_X^{N,n}$ from which are drawn the points $(X_i^l, w_i^l)_{i\in\{1,..,n\}, l\in\{1,..,K\}}$. Several remarks have to be made at this stage:

- first, it is not clear whether the quadrature weights and points $(w_k^l, X_k^l)_{k\in\{1,...n\}}$ are a consistent approximation of $(w_i, X_i)_{i\in\{1,...,N\}}$. If $w_i = \frac{1}{N}, \forall i \in \{1, ..., N\}$ and $w_k^l = \frac{1}{n}, \forall k \in \{1, .., n\}, l \in \{1, .., K\}$ then it is enough. In the general case, this is not obvious.
- Of course, the term $\mathcal{O}(N^\beta)$ is usually dropped (i.e. neglected) or, in a sense, is contained in the generalisation gap, see section 1.
- Now assume the quadrature weights are as in the previous points, then we can asymptotically characterise the MC error in (36): introduce an arbitrary probability measure $\mathrm{d}\mathcal{P}_X^{N,n}$ with only constraint that asymptotically with the number of batches $K = \frac{N}{n}$ we have[18] [19]

$$\int L'(X, \theta_j) \nabla_\theta u(X, \theta_j) \, \mathrm{d}\mathcal{P}_X^{N,n} \underset{K\sim\infty}{=} \frac{1}{N} \sum_{k=1}^{N} L'(X_k, \theta_j) \nabla_\theta u(X_k, \theta_j).$$

[17] different convergence rates for different experimental designs, see [60] and above all the references therein.
[18] For example, the following probability measure has the above properties:

- introduce
$$\mathrm{d}\mathcal{P}_{\mathcal{S}}(x) = \sum_{i\in\{X_1,...,X_N\}\setminus\mathcal{S}} \frac{1}{Card(\{X_1, ..., X_N\} \setminus \mathcal{S})} \delta_{X_i}(x), \text{ where } \mathcal{S} \text{ is a set.}$$
- Then we can build
$$\mathrm{d}\mathcal{P}(x_1, ..., x_N) = \mathrm{d}\mathcal{P}_{\{\emptyset\}}(x_1) \, \mathrm{d}\mathcal{P}_{\{x_1\}}(x_2) \, \mathrm{d}\mathcal{P}_{\{x_1,x_2\}}(x_3) ... \, \mathrm{d}\mathcal{P}_{\{x_1,x_2,...,x_{N-1}\}}(x_N),$$
and K groups of one sampling $((X_1^l, ..., X_n^l))_{l\in\{1,...,K\}}$ from the above probability measure.

[19] Note that $\mathrm{d}\mathcal{P}_X$ is also a good candidate as it satisfies the conditions.

Independently of the probability measure satisfying the above conditions, we have

$$\begin{aligned}\int L'(X,\theta_j^l)\nabla_\theta u(X,\theta_j^l)\,\mathrm{d}\mathcal{P}_X^{N,n} &= \frac{1}{N}\sum_{k=1}^{N} L'(X_k,\theta_j)\nabla_\theta u(X_k,\theta_j) + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right),\\ &= \frac{1}{N}\sum_{k=1}^{N} L'(X_k,\theta_j^l)\nabla_\theta u(X_k,\theta_j^l) + \frac{\sigma_j^l}{\sqrt{n}}\mathcal{G}^l,\end{aligned}$$

where $\mathcal{G}^l$ is a vector of independent gaussian random variables $(\mathcal{G}_1^l,...,\mathcal{G}_{Card(\theta)}^l)^T$, where $\forall i \in \{1,...,Card(\theta)\}, \forall l \in \{1,..,K\}, \mathcal{G}_i^l \sim \mathcal{G}(0,1)$ with zero mean, unitary variance. Furthermore, $[\sigma\sigma^T]_j^l \in \mathbb{R}^{Card(\theta)\times Card(\theta)}$ is a symmetric matrix[20] given by

$$\begin{aligned}[\sigma\sigma^T]_j^l &= \int (L'(X,\theta_j^l))^2\nabla_\theta u(X,\theta_j^l)\nabla_\theta^T u(X,\theta_j^l)\,\mathrm{d}\mathcal{P}_X^{N,n}\\ &\qquad - \left[\int L'(X,\theta_j^l)\nabla_\theta u(X,\theta_j^l)\,\mathrm{d}\mathcal{P}_X^{N,n}\right]\left[\int L'(X,\theta_j^l)\nabla_\theta^T u(X,\theta_j^l)\,\mathrm{d}\mathcal{P}_X^{N,n}\right],\\ &= \frac{1}{n}\sum_{k=1}^{n}(L'(X_k^l,\theta_j^l))^2\nabla_\theta u(X_k^l,\theta_j^l)\nabla_\theta^T u(X_k^l,\theta_j^l)\\ &\qquad - \frac{1}{n^2}\sum_{k,p=1}^{n} L'(X_k^l,\theta_j^l)\nabla_\theta u(X_k^l,\theta_j^l)L'(X_p^l,\theta_j^l)\nabla_\theta^T u(X_p^l,\theta_j^l).\end{aligned}$$

For one batch, we consequently have

$$\theta_{j+1}^{l+1} = \theta_j^l - \frac{\gamma}{N}\sum_{k=1}^{N} L'(X_k^l,\theta_j^l)\nabla_\theta u(X_k^l,\theta_j^l) + \frac{\gamma}{\sqrt{n}}\sigma_j^l\mathcal{G}^l. \tag{37}$$

Remark 3.6 *We suggest here doing the same as in remark 3.5, where we did some choices in term of $\alpha, \theta^* - \theta$ in order to rewrite GD in a PDE resolution framework, but for* one batch of SGD. *If we choose,*

- *$\alpha = -\nabla_\theta u(X,\theta)$, we have the good first term in the drift.*
- *Due to the previous choice, the second term in the drift becomes*

$$\int L''(X,\theta)\nabla_\theta u(X,\theta)\nabla_\theta^T u(X,\theta)(\theta^* - \theta)\,\mathrm{d}\mathcal{P}_X,$$

  so that only remains $\theta^* - \theta$ as degree of freedom. In (37), the second term does not appear so
  - *either $\theta^* - \theta = 0$ but this forbids some stochasticity afterward,*
  - *or $L'' = 0$ and this implies $L' = cste$ and L is close to a L^1-norm.*
- *For the above reason, let us assume that L is a (smooth[21]) L^1-norm,*
- *then we can work on the choice of $\theta^* - \theta$ to revisit SGD in our PDE framework. The general term of matrix $[\sigma\sigma^T]_j^l$ is*

$$\begin{aligned}\left[[\sigma\sigma^T]_j^l\right]_{p,q} &= \int (L'(X,\theta_j))^2\partial_{\theta_p}u(X,\theta_j^l)\partial_{\theta_q}u(X,\theta_j^l)\,\mathrm{d}\mathcal{P}_X^{N,n}\\ &\qquad - \int L'(X,\theta_j)\partial_{\theta_p}u(X,\theta_j^l)\,\mathrm{d}\mathcal{P}_X^{N,n}\int L'(X,\theta_j)\partial_{\theta_q}u(X,\theta_j^l)\,\mathrm{d}\mathcal{P}_X^{N,n},\end{aligned}$$

[20] Note that the results of these few lines are independent of the choice of $\mathrm{d}\mathcal{P}_X^{N,n}$.

[21] To avoid singular points where L'' would not be zero. The attentive reader may now have a clue why we chose $L(x,y) = x - y$ in the *fil rouge* problem.

which can be compared to (21)

$$\begin{aligned}\int \left[\sigma\sigma^T\right]_{p,q} \mathrm{d}\mathcal{P}_X &= \int 2L'(X,\theta)\alpha_p(\theta_q^* - \theta_q)\,\mathrm{d}\mathcal{P}_X,\\ &= \int 2L'(X,\theta)\nabla_{\theta_p}u(X,\theta_j^l)(\theta_q^* - \theta_q)\,\mathrm{d}\mathcal{P}_X.\end{aligned}$$

If we choose

$$\theta^* - \theta = \frac{1}{2}L'(X,\theta_j^l)\nabla_\theta^T u(X,\theta_j^l) - \frac{1}{2}\int L'(X,\theta_j^l)\nabla_\theta^T u(X,\theta_j^l)\,\mathrm{d}\mathcal{P}_X^{N,n}, \tag{38}$$

and $\mathrm{d}\mathcal{P}_X^{N,n} \sim \mathrm{d}\mathcal{P}_X$[22] *then we recover the general term of* $[\sigma\sigma^T]_j^l$.

Once again, we can recast SGD in a PDE resolution framework by making some particular choices for $\alpha, \theta^* - \theta$, L *and* $\mathrm{d}\mathcal{P}_X^{N,n}$ *(and* $\hat{u} \equiv 0$*).*

With remark 3.6, we revisited SGD as a combination of choices and numerical methods which allows rewriting SGD in a PDE resolution framework. We can also revisit the reverse engineering we performed earlier for GD: asymptotically as $\gamma = \Delta t$ goes to zero, SGD solves

$$\begin{aligned}0 = \;& \partial_t\overline{u}(t,\theta)\\ & - \sum_{i=1}^{Card(\theta)} \partial_{\theta_i}\overline{u}(t,\theta)\int \left[\partial_{\theta_i}u(t,X,\theta)L'(t,X,\theta)\right]\mathrm{d}\mathcal{P}_X\\ & + \frac{1}{2}\sum_{i,j=1}^{Card(\theta)} \partial^2_{\theta_i,\theta_j}\overline{u}(t,\theta)\int \begin{bmatrix} (L'(t,X,\theta))^2\partial_{\theta_i}u(t,X,\theta)\partial_{\theta_j}u(t,X,\theta) \\ -L'(t,X,\theta)\partial_{\theta_i}u(t,X,\theta)\displaystyle\int L'(t,X,\theta)\partial_{\theta_j}u(t,X,\theta)\,\mathrm{d}\mathcal{P}_X^{N,n}\end{bmatrix}\mathrm{d}\mathcal{P}_X,\\ & \text{with } \overline{u}(t,\theta) = \int u(t,X,\theta)\,\mathrm{d}\mathcal{P}_X.\end{aligned} \tag{39}$$

Furthermore, if $\mathrm{d}\mathcal{P}_X^{N,n} \sim \mathrm{d}\mathcal{P}_X$, then (39) resumes to (remember we must have $L'' = 0$)

$$\begin{aligned}0 = \;& \partial_t\overline{u}(t,\theta)\\ & - \sum_{i=1}^{Card(\theta)} \partial_{\theta_i}\overline{u}(t,\theta)\int \left[\partial_{\theta_i}u(t,X,\theta)L'(t,X,\theta)\right]\mathrm{d}\mathcal{P}_X\\ & + \frac{1}{2}\sum_{i,j=1}^{Card(\theta)} \partial^2_{\theta_i,\theta_j}\overline{u}(t,\theta)\begin{bmatrix} +\displaystyle\int (L'(t,X,\theta))^2\partial_{\theta_i}u(t,X,\theta)\partial_{\theta_j}u(t,X,\theta)\,\mathrm{d}\mathcal{P}_X \\ -\displaystyle\int L'(t,X,\theta)\partial_{\theta_i}u(t,X,\theta)\,\mathrm{d}\mathcal{P}_X\int L'(t,X,\theta)\partial_{\theta_j}u(t,X,\theta)\,\mathrm{d}\mathcal{P}_X\end{bmatrix},\\ & \text{with } \overline{u}(t,\theta) = \int u(t,X,\theta)\,\mathrm{d}\mathcal{P}_X.\end{aligned} \tag{40}$$

With remark 3.6, we see that the consistency of SGD in order to solve the desired PDE (hence our minimisation) depends on L. Now, once again, under some conditions, the solution of (27) may coincide with the solution of (40): assume for example that $\alpha = -\nabla_\theta u$, $\theta^* - \theta$ as in (38) and $\mathrm{d}\mathcal{P}_X^{N,n} \sim \mathrm{d}\mathcal{P}_X$ in (27) together with

1. $L'' \sim \delta \sim 0$,
2. $\hat{u} \sim \delta \sim 0$,

then (27) degenerates toward (40). In this case, solving (40) is equivalent to solving (27) in the regime $\delta \sim 0$. We will come back to such analysis, more rigorously, in section 4. Note that conditions of adequacy for GD have been put forward in the previous lines of remark 3.5: from the two above points for SGD, we can see that the adequacy conditions for SGD are less stringent than the ones for GD, (cf. point 3. of remark 3.5 for GD for example).

[22] which is not always the case in SGD algorithms.

Remark 3.7 *Note that many other optimisation algorithms exist, see [65] for an overview. We can not go through every of them in this document and study their conditional consistency.*

Let us now revisit example 3.1.1 from the SGD point of view.

Example 3.5.1 (Stochastic Gradient Descent (SGD) as in ML optimisers) *We here suggest revisiting the* fil rouge *problem of examples 3.1.1–3.1.3 with the stochastic GD (SGD) algorithm presented above. Once again, we assume that we do not have access to the hessian of the functional we aim at minimising and we introduce some stochasticity by using minibatches. Recall, see example 2.0.1, that the functional we minimise in our example can be related to a particular architecture with two points $N = 2$ in the training set: $(X_1 = 0, X_2 = 6)$ such that $(u(X_1) = 0, u(X_2) = 0)$. As a consequence, in this case, the only possibility here is to consider minibatches of size $n = 1$. We here choose $\mathrm{d}\mathcal{P}_X^{N,n}$ to be $\mathrm{d}\mathcal{P}_X$.*

Figure 4 presents the results obtained with SGD. The conditions are similar to the ones of figures 1–2 in term of initial conditions and learning rates. For the first line of figure 4, the initial point is $\theta_0 = -5$ and the learning rate is $\gamma = 3.5$. The results are close to the ones obtained with GD because the stochasticity induced by the use of minibatching is not important enough to significantly change the results. Still, on the right column of the first line, we can see that the process slightly oscillates around the value $\theta^* = 0$: those oscillations are due to the introduction of a stochastic ingredient and are not related to any numerical instabilities as in figure 3 for example. The second line of figure 4 shows the results obtained for $\theta_0 = 11$ and $\gamma = 3.5$. It must be compared to the third line of figure 2 with GD: the introduction of stochasticity via *minibatching prevents from remaining stuck in a cycle in this case as in figure 2. The process $n \to \theta_n$ comes closer to the minimum of functional J. Once again, the oscillations we can observe on the right picture of this same second line of figure 4 are only due to the introduction of stochasticity (and not numerical instabilities). Note that in this case,* the last point • is not necessarily the best point •. *In this sense, SGD is more an* exploration algorithm than an optimisation algorithm. *Now, assume that we keep in memory the point $(\theta_n)_{n\in\{1,\dots,n_{\text{epoch}}=2000\}}$ having the smallest error (cf. the discussion of section 3.3 about the instrumention of the simulation code solving a PDE for our ML problem), then the results obtained with SGD are way better than for GD in this case.*

The last line of figure 4 finally shows that if the learning rate is not finely enough tuned, the process $n \to \theta_n$ may jump in regions where the gradient is zero and remain stuck there, just as for GD. Once again, the last point is not necessarily the best point.

Finally, of course, the computation for SGD took $\frac{N}{n}$ (here twice) as much time as for GD. In this case $\frac{N}{n} = 2$ and one epoch/iteration stands for $\frac{N}{n} = 2$ time steps.

Example 3.5.2 (SGD as in ML optimisers, quantitative results) *The results of example 3.5.1 related to figure 4 were mainly qualitative. We here suggest some quantitative ones.*

The conditions are the same as in the previous (quantitative) examples 3.1.2–3.1.4. The probabilities for SGD algorithm to reach the vicinities of the local/global minima are displayed in table 4 for several values of the learning rate. The SGD results of table 4 can be compared to GD results of table 2:

- *for $\gamma = 0.05$, SGD does not lead to significant improvements as attests the confidence intervals for $\gamma = 0.05$ in tables 2–4.*
- *for $\gamma = 3.50$, SGD does*

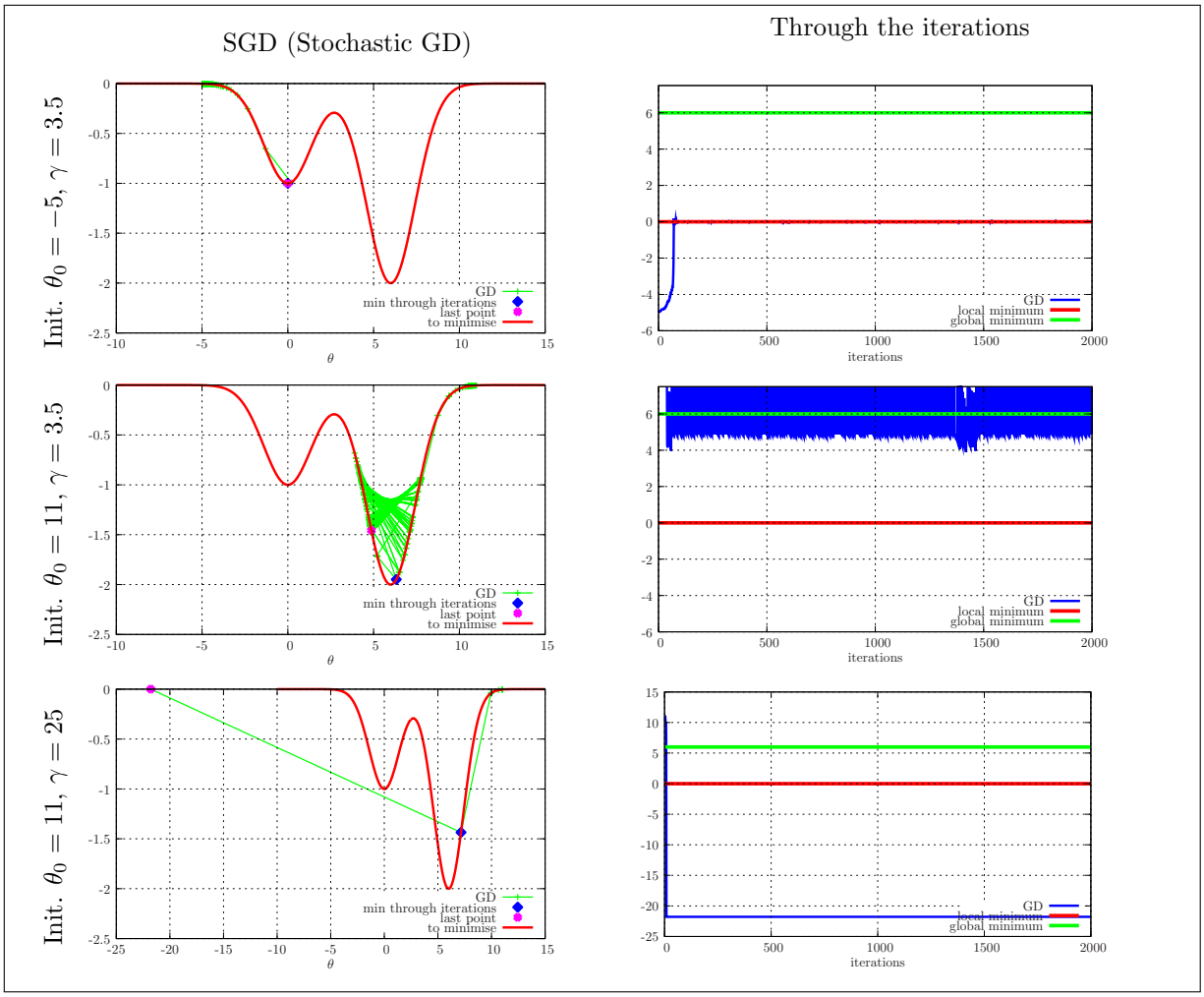

Figure 4: These figures revisit the test-case of example 3.1.1 but with a SGD algorithm with learning rates $\gamma = 3.5, 25.0$. In this case, the number of points $N = 2$, given by $(0, 6)$ and the batch size can only be $n = 1$. Except from the type of algorithm, the condition are similar to the ones of figures 2–1 in term of initial points or of learning rates. Note that the computations took twice the same amount of time as for GD.

- *preserve the behaviour of GD in the vicinity of $\theta^* = 0$,*
- *lead to statistically significative (cf. the confidence intervals for $\gamma = 3.5$ in tables 2–4) but small improvements in the vicinity of $\theta^* = 6$: from less than $\approx 0.31\%$ to a little more than 0.705%.*

– *for $\gamma = 25.0$, it is not clear whether SGD provides better results than GD due to the intersecting confidence intervals (cf. the confidence intervals for $\gamma = 25.0$ in tables 2–4).*

The last line of table 4 presents the probabilities for SGD to recover the narrower ($\epsilon = 10^{-4}$) vicinities of the local/global minima: SGD does slightly improve the results with respect to GD (cf. last line of table 2) in this case.

Initialiser $\theta_0 \sim \mathcal{U}([-15,15])$	Vicinity of $\theta^* = 0$	Vicinity of $\theta^* = 6$
SGD ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10^{-2}, 10^{-2}])$	$\mathbb{P}(\theta_{\min} \in [6-10^{-2}, 6+10^{-2}])$
$\gamma = 0.05$	$25.4\% \in [24.54\%, 26.25\%]$	$28.0\% \in [27.10\%, 28.86\%]$
$\gamma = 3.50$	$32.1\% \in [31.15\%, 32.99\%]$	$0.89\% \in [0.705\%, 1.075\%]$
$\gamma = 25.0$	$0.27\% \in [0.168\%, 0.372\%]$	$0.29\% \in [0.184\%, 0.396\%]$
SGD ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10^{-4}, 10^{-4}])$	$\mathbb{P}(\theta_{\min} \in [6-10^{-4}, 6+10^{-4}])$
$\gamma = 0.05$	$0.01\% \in [0.00\%, 0.03\%]$	$0.02\% \in [0.00\%, 0.05\%]$

Table 4: Probabilities for the Stochastic GD (SGD) algorithm to recover the local ($\theta^* = 0$) or the global ($\theta^* = 6$) minimum of (4) for several learning rates γ. To compute the probabilities, we have resort to $N_{seed} = 10^4$ multiple initialisations with $\theta_0 \sim \mathcal{U}([-15,15])$. Confidence intervals for the results are provided. Comments are provided in example 3.5.2.

In the next section, we suggest applying a consistent discretisation of (27) (neglecting the terms with $\hat{u}$ just as classically done in ML algorithms) given by (32) and revisit the *fil rouge* example of this paper.

3.6 A consistent optimiser inspired from a stochastic PDE resolutions

In the previous section, we chose $\alpha, \theta^* - \theta, L$ and even $\mathrm{d}\mathcal{P}_X^{N,n}$ (and assumed $\hat{u} \equiv 0$) in expression

$$\begin{aligned} \overline{\theta}_{n+1} = \overline{\theta}_n \quad &+ \int \left[\alpha L'(X, \overline{\theta}_n) + L''(X, \overline{\theta}_n)\alpha \nabla_\theta^T u_{ANN}(X, \overline{\theta}_n)(\theta^* - \overline{\theta}_n)\right] \mathrm{d}\mathcal{P}_X^{N,n} \\ &+ \sqrt{\Delta t}\left[\int \sigma(X, \alpha, \theta^* - \overline{\theta}_n)\, \mathrm{d}\mathcal{P}_X^{N,n}\right] \mathcal{G}, \end{aligned} \tag{41}$$

which recovers classical ML optimiser algorithms. From (41), we were able to build the PDE the algorithm asymptotically solves, in the general case. It asymptotically solves[23]

$$\begin{aligned} 0 = \quad & \partial_t \overline{u}(t,\theta) \\ &+ \sum_{i=1}^{Card(\theta)} \partial_{\theta_i} \overline{u}(t,\theta) \int \left[\alpha_i L'(t,X,\theta) + L''(t,X,\theta)\alpha_i \sum_{j=1}^{Card(\theta)} \partial_{\theta_j} u(t,X,\theta)(\theta_j^* - \theta_j)\right] \mathrm{d}\mathcal{P}_X \\ &+ \frac{1}{2} \sum_{i,j=1}^{Card(\theta)} \partial^2_{\theta_i,\theta_j} \overline{u}(t,\theta) \int \left[2L'(t,X,\theta)\alpha_i(\theta_j^* - \theta_j)\right] \mathrm{d}\mathcal{P}_X, \end{aligned} \tag{42}$$

when the time step (learning rate) goes to zero and the number of simulated stochastic processes goes to infinity.

We here suggest making some assumptions on $\alpha, \theta^* - \theta$, $\mathrm{d}\mathcal{P}_X^{N,n}$ (and still $\hat{u} \equiv 0$ for the moment), different from the ones usually made for ML optimisers (GD or SGD), which allows (41) to remain consistent with (42).

First, the choices for $\alpha, \theta^* - \theta$, $\mathrm{d}\mathcal{P}_X^{N,n}$ (and still $\hat{u} \equiv 0$ for the moment) are not unique, as already testified the examples of the previous section with GD and SGD. Let us begin by one particular choice, some others will be briefly studied afterward.

3.6.1 Particular choices for $\alpha, \theta^* - \theta$ generalising SGD (and what we gain)

Let us choose

- $\alpha = -\nabla_\theta u(X, \theta)$ as for GD and SGD,

[23] (42) corresponds to (27) while cancelling the terms with $\hat{u}$.

– $\theta^* - \theta = \Sigma^2(\theta)$ with $\theta \to \Sigma(\theta)$ arbitrary at this stage of the discussion,

– $\mathrm{d}\mathcal{P}_X^{N,n} \sim \mathrm{d}\mathcal{P}_X^N$.

In this particular case, (41) and (42) become

$$\begin{aligned}\overline{\theta}_{n+1} = \overline{\theta}_n \quad & -\frac{\Delta t}{N}\sum_{i=1}^{N}\left[\nabla_\theta u(X,\overline{\theta}_n)L'(X_i,\overline{\theta}_n) + L''(X_i,\overline{\theta}_n)\nabla_\theta u(X,\overline{\theta}_n)\nabla_\theta^T u_{ANN}(X_i,\overline{\theta}_n)\Sigma(\overline{\theta}_n)\right] \\ & +\frac{\sqrt{\Delta t}}{N}\left[\sum_{i=1}^{N}\sqrt{|L'(X_i,\theta_n)\nabla_\theta u(X_i,\theta_n)|\Sigma(\theta_n)}\right]\mathcal{G},\end{aligned} \tag{43}$$

which solves

$$\begin{aligned}0 = \quad & \partial_t\overline{u}(t,\theta) \\ & -\sum_{i=1}^{Card(\theta)}\partial_{\theta_i}\overline{u}(t,\theta)\underbrace{\int\left[\nabla_{\theta_i}u(t,X,\theta)L'(t,X,\theta) + L''(t,X,\theta)\nabla_{\theta_i}u(X,\theta)\sum_{j=1}^{Card(\theta)}\partial_{\theta_j}u(t,X,\theta)\Sigma_j(\theta)\right]\mathrm{d}\mathcal{P}_X}_{\mu_i(t,\theta)} \\ & +\frac{1}{2}\sum_{i,j=1}^{Card(\theta)}\partial^2_{\theta_i,\theta_j}\overline{u}(t,\theta)\underbrace{\int 2|L'(t,X,\theta)\nabla_{\theta_i}u(X,\theta)|\Sigma_j(\theta)\,\mathrm{d}\mathcal{P}_X}_{[\sigma(t,\theta)\sigma^T(t,\theta)]_{i,j}},\end{aligned} \tag{44}$$

provided a small enough time step. The question now is: what is the gain with respect to the choices made with GD and SGD? The solution of (44) now coincides with the solution of (27) under even less stringent conditions as SGD: we only need to have

1. $\hat{u} \sim \delta \sim 0$.

Furthermore, classical results from ODE[24]/PDE[25] theory state that taking a time step at iteration n of the form

$$\Delta t^n = CFL \times \epsilon \times \min\left(\frac{1}{\max\limits_{i\in\{1,\dots,Card(\theta)\}}\mu_i(t^n,\overline{\theta}_n)}, \frac{1}{\max\limits_{i\in\{1,\dots,Card(\theta)\}}\sigma_i(t^n,\overline{\theta}_n)}\right), \tag{45}$$

ensures having stability if $CFL \leq 1$ and an accuracy $\mathcal{O}(\epsilon)$ for process $(\overline{\theta}_n)_{n\in\mathbb{N}}$ to approximate $(\overline{\theta}_t)_{t\in\mathbb{R}^+}$. Note that the above upper bound for the time step is not optimal: it is obtained by analysing separatedly the drift regime and the diffusion one and taking the most constraining value. More elaborate time step criterion could lead to better efficiencies (same accuracy with bigger time steps). Still, let us test this simple choice for the minimisation of functional (4) in example 3.6.1.

Example 3.6.1 (A consistent SGD as in a PDE framework) *In this example, we once again revisit the test-case of examples 2.0.1–3.1.3–3.5.1 but with the new PDE consistent solver discretised with an MC scheme. We make particular choices of $\alpha, \theta^* - \theta$. For this example, (29) resumes to*

$$\begin{cases} 0 = \quad \partial_t\overline{u}(t,\theta) + \partial_\theta\overline{u}(t,\theta)\displaystyle\int\left[\alpha L'(t,X,\theta) + L''(t,X,\theta)\alpha\partial_\theta u(t,X,\theta)(\theta^*-\theta)\right]\mathrm{d}\mathcal{P}_X \\ \qquad\quad +\dfrac{1}{2}\partial^2_{\theta,\theta}\overline{u}(t,\theta)\displaystyle\int\left[2L'(t,X,\theta)\alpha(\theta^*-\theta)\right]\mathrm{d}\mathcal{P}_X, \forall t\in[0,T], \forall\theta\in\Theta, \\ \overline{u}(t=0,\theta) = \overline{u}^0(\theta), \forall\theta\in\Theta, \\ \text{together with looking for } t^*,\theta^* \text{ such that } J(\overline{u}(t^*,\theta^*)) \leq J(\overline{u}(t,\theta)), \forall t\in[0,T], \theta\in\Theta. \end{cases} \tag{46}$$

[24] as the equation on $\overline{\theta}_t$ is an ODE.

[25] As the equation satisfied by $\mathbb{E}[u^0(\overline{\theta}_t^\theta)]$ is a PDE.

First, in example 2.0.1, we do not have a test-set so that in this example we take $\overline{J} = J$, cf. last line of (46). Let us now choose

- *$L(x,y) = (x-y)$ as defined in example 2.0.1 (so that $L'(x,y) = -1, L''(x,y) = 0$),*
- *$\alpha = -\partial_\theta u(X,\theta)$ as for GD and SGD, with $X, \theta \to u(X,\theta)$ as defined in example 2.0.1.*
- *Let us choose furthermore $\mathrm{d}\mathcal{P}_X \sim \mathrm{d}\mathcal{P}_X^N$ with $\mathrm{d}\mathcal{P}_X^N$ as defined in example 2.0.1.*
- *With the above choices, we have $\int \alpha L' \,\mathrm{d}\mathcal{P}_X = -\int L' \nabla_\theta u \,\mathrm{d}\mathcal{P}_X = -J'(\theta)$ with J given by (4).*
- *It remains to choose $\theta^* - \theta$: the diffusion coefficient is $\alpha L'(\theta^* - \theta)$ integrated over $\mathrm{d}\mathcal{P}_X$. Let us choose it such that*

$$\int 2\alpha L'(\theta^* - \theta) \,\mathrm{d}\mathcal{P}_X = |J'(\theta)|\Sigma^2, \text{ with } \Sigma = cste.$$

  This leads to taking $(\theta^* - \theta) = \Sigma(\theta) = -\frac{1}{2}\Sigma^2 \frac{1}{J'(\theta)}$. This ensures

  - *having a wellposed drift-diffusion problem from the PDE point of view,*
  - *going in the opposite direction of the gradient from an ML point of view.*

Finally, with the above choices, we aim at solving

$$\begin{cases} \partial_t \overline{u}(t,\theta) - J'(\theta)\nabla_\theta \overline{u}(t,\theta) + \dfrac{1}{2}|J'(\theta)|\Sigma^2 \nabla^2_{\theta,\theta}\overline{u}(t,\theta) = 0, \forall t \in [0,T], \forall \theta \in \Theta, \\ u^0(\theta) = \overline{u}(t=0,\theta), \forall \theta \in \Theta, \\ \text{together with looking for } t^*, \theta^* \text{ such that } J(t^*,\theta^*) \leq J(t,\theta) \ \forall t \in [0,T], \theta \in \Theta. \end{cases} \tag{47}$$

In (47), due to the conditions of the different fil rouge *examples of this document,*

- *we have $\Theta = \mathbb{R}$,*
- *T is some final time which remains to be defined. It is given by $T = n_{\text{epoch}} \times \gamma$ for a fixed learning rate γ and the choice of n_{epoch} iterations/epochs.*
- *$u^0(\theta)$ is the initial condition, related to the choice of the initialiser. In the following, depending on the studies/figures, we choose:*
  - *$u^0(\theta) = \delta_{\theta_0}(\theta)$ with $\theta_0 = -5$ and $\theta_0 = 11$, cf. the results of figures 5–6,*
  - *or, later on in example 3.6.2, we take $u^0(\theta) = \mathbf{1}_{[-15,15]}(\theta)$ corresponding to initialiser $\theta_0 \sim \mathcal{U}([-15,15])$.*
- *Finally $J(t,\theta) = J(\overline{u}(t,\theta))$ must be computed during the resolution of the PDE.*

Now, in order to solve (47), we can rely on several numerical methods. As we aim at being able to tackle high dimensional problems (even if here, θ is scalar), we choose a numerical scheme whose convergence rate is independent of the dimension, i.e. an MC scheme. For this, we rely on both Ito's lemma 1 and Kolmogorov's theorem 2: averaging over realisations of the stochastic process given by

$$\mathrm{d}\overline{\theta}_t = -J'(\overline{\theta}_t)\,\mathrm{d}t + \sqrt{|J'(\overline{\theta}_t)|}\Sigma(\overline{\theta}_t)\,\mathrm{d}W_t, \tag{48}$$

ensures solving (47) as $\overline{u}(t,\theta) = \mathbb{E}[u^0(\overline{\theta}_t^{\theta_0})]$ with θ^0 sampled according to the initialiser. In order to instrument the resolution, it is enough evaluating $J(t,\theta) = J(t,\overline{\theta}_t = \theta) = J(\overline{\theta}_t = \theta)$, and keeping

in memory $\theta^* = \overline{\theta}_{t^*}$ *realising the minimum.*
Now, (48) *can not, in general, be solved analytically. Let us discretise* (48) *thanks to an explicit Euler scheme of learning rate/time step* $\gamma = \Delta t$*, then solving* (47) *resumes to averaging over the discrete stochastic process given by*

$$\overline{\theta}_{n+1} = \overline{\theta}_n - \Delta t J'(\overline{\theta}_n) + \sqrt{\Delta t |J'(\overline{\theta}_n)|}\mathcal{G}(0, \Sigma^2(\overline{\theta}_n)). \tag{49}$$

Besides, $J(\overline{\theta}_{t^*}) \approx J(\overline{\theta}_{n^*})$ *where* n^* *is defined by* $J(\overline{\theta}_{n^*}) \leq J(\overline{\theta}_n)\ \forall n \in [0, N_T]$ *where* N_T *is such that* $\sum_{n=0}^{N_T} \Delta t_n = T$*. For constant learning rate/time step* $\Delta t = \gamma$*,* $T = n_{\text{epoch}} \times \gamma$ *where* n_{epoch} *is the number of iterations/epochs. In the next computations of figure 5 for example, we take* $T = n_{\text{epoch}} \times \gamma = 2000 \times 3.5 = 7000$*. Now, stochastic process* (49) *is not harder to simulate than an SGD one. We can discretise it and instrument its path in order to solve our minimisation of functional* (4)*, see figure 5. Note also that the PDE framework and the fact we use an explicit Euler scheme allows proving that the optimiser error will be* $J(\overline{\theta}_{t^*}) = J(\overline{\theta}_{n^*}) + \mathcal{O}(\sqrt{\Delta t})$*. Relying on this PDE framework allows quantifying the error made during the discretisation of the optimisation phase, see the discussion of section 1.*

The first line of figure 5 presents results obtained simulating (49) *with initialiser* $\delta_{\theta_0}(\theta)$ *with* $\theta_0 = -5$*,* $\gamma = \Delta t = 3.5$ *and* $\Sigma = 1$*: the discretised stochastic process allows jumping from one local minimum to the other and explores the whole space of the weight* $\theta \in \Theta = [-15, 15]$*. Even if beginning at* $\theta_0 = -5$*, the process comes close to* $\theta^* = 6$*. Of course, once again, the stochastic process explores more than optimises and the last point* • *is not necessarily the best point* •*. One needs to keep in memory the set of weights having the lowest error during the exploration. On the right column of the first line, we can see that the fluctuations of the process switch between the vicinities of* $\theta^* = 0$ *and* $\theta^* = 6$*.*

The second line of figure 5 presents the same results but with $\theta_0 = 11$*. Once again, the process explores the whole space* Θ*. What is interesting with such stochastic process is that the whole space is explored with only one initialisation. In this simple example, the number of minima is only* 2 *but classical ML problems are known to have exponentially many local minima [4] with the dimension* d_{in} *and the number of training points* N*. Being able to explore several of them with the same initialisation/stochastic process can be precious.*

Of course, here, we have $\Sigma = 1$*: this quantity pilots the stochasticity and allows (or not) jumping from one local minima to the other. For example, in the case of SGD as in example 3.5.1 and figure 4, the stochasticity induced by minibatching was not enough to jump (and the only possibility was* 1 *minibatch, cf. example 3.5.1). In a same manner, choosing* Σ *too small would lead to the same results as SGD. We suggest postponing the discussion on the relevant choice of* Σ *to another section/example, we just want here to insist on the fact that the PDE framework allows being more general than SGD.*

The two previous examples do not fully take advantage of the PDE framework we just worked on: from this framework, we can derive stability conditions (to avoid spurious oscillations as in the pictures of figure 3) and even accuracy conditions to make sure the error for the PDE resolution remains proportional to a certain quantity. Let us detail this point now. Those are classical PDE results [8]: equation (47) *can be decomposed into two regimes, the advective one and the diffusion one. To be stable and accurate with accuracy* $\mathcal{O}(\varepsilon)$ *in the advective regime, it is enough choosing* Δt *proportional to* $CFL \times \frac{\varepsilon}{|J'(\theta)|}$ *with* $CFL \leq 1$*. To be stable and accurate with accuracy* $\mathcal{O}(\varepsilon)$ *in the diffusion regime, it is enough having* Δt *proportional to* $CFL \times \frac{\varepsilon}{\sqrt{|J'(\theta)||\Sigma(\theta)|}}$*. If now we choose (particular case of* (45)*)*

$$\Delta t = CFL \times \varepsilon \times \min\left(\frac{1}{|J'(\theta)|}, \frac{1}{\sqrt{|J'(\theta)||\Sigma(\theta)|}}\right), \tag{50}$$

the algorithm should be stable and $\mathcal{O}(\varepsilon)$ provided $CFL \leq 1$. Of course, the above strategy is way less easy to apply in high dimension and needs a (parallel) reduction to compute the minimum over the whole set of components of the gradient. *Still, let us see what can be expected from such strategy. The results obtained with the PDE inspired SGD together with the above time step strategy are displayed in figure 6: once again, the figures are obtained in the same conditions as the previous ones but with this new time step/learning rate strategy. Independently of the starting point, the stochastic process explores the whole space. Much less time is spent jumping from one side to the other. Also, the noise, related to the error, is more homogeneous.*

The last line of figure 6 must be compared to the second line of figure 3 for which numerical instabilities made GD explode toward infinity. With the time step/learning rate strategy (50), the stochastic process is stable and gives very good results. We insist on the fact that stability/accuracy conditions can theoretically be derived for any choices of $\alpha, \theta^* - \theta$ (i.e. even for SGD as in section 3.5 for example) once characterised.

Furthermore, the computations took exactly the same amount of time as for GD with this PDE inspired SGD (instead of $\times \frac{N}{n}$ for SGD).

Finally, I hope the reader now understands why we chose such a simple functional (4): classical SGD cannot be consistent if $L''(x, y) \neq 0$ so we at least wanted to be able to compare the PDE inspired SGD to SGD in favorable conditions for SGD.

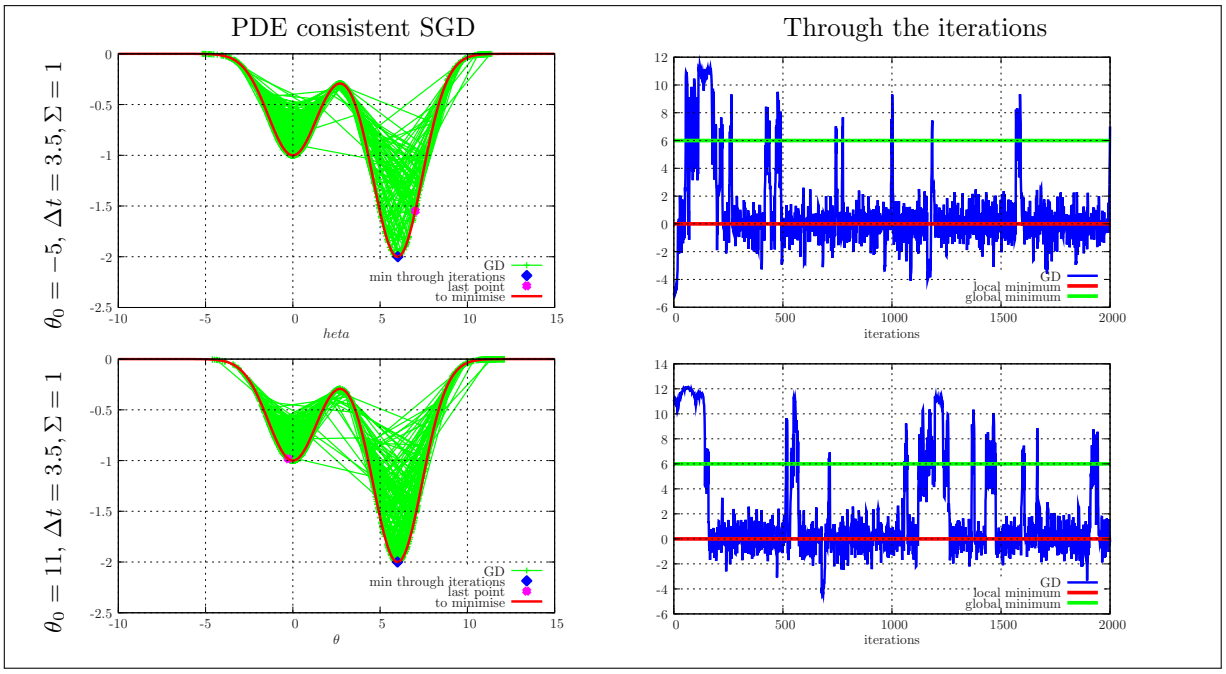

Figure 5: These figures revisit the test-case of example 3.1.1 but with a PDE consistent SGD algorithm with time step $\Delta t = \gamma = 3.5$. In this case, the stochasticity does not come from minibatching. Except from the type of algorithm, the condition are similar to the ones of figures 2–1 in term of initial points.

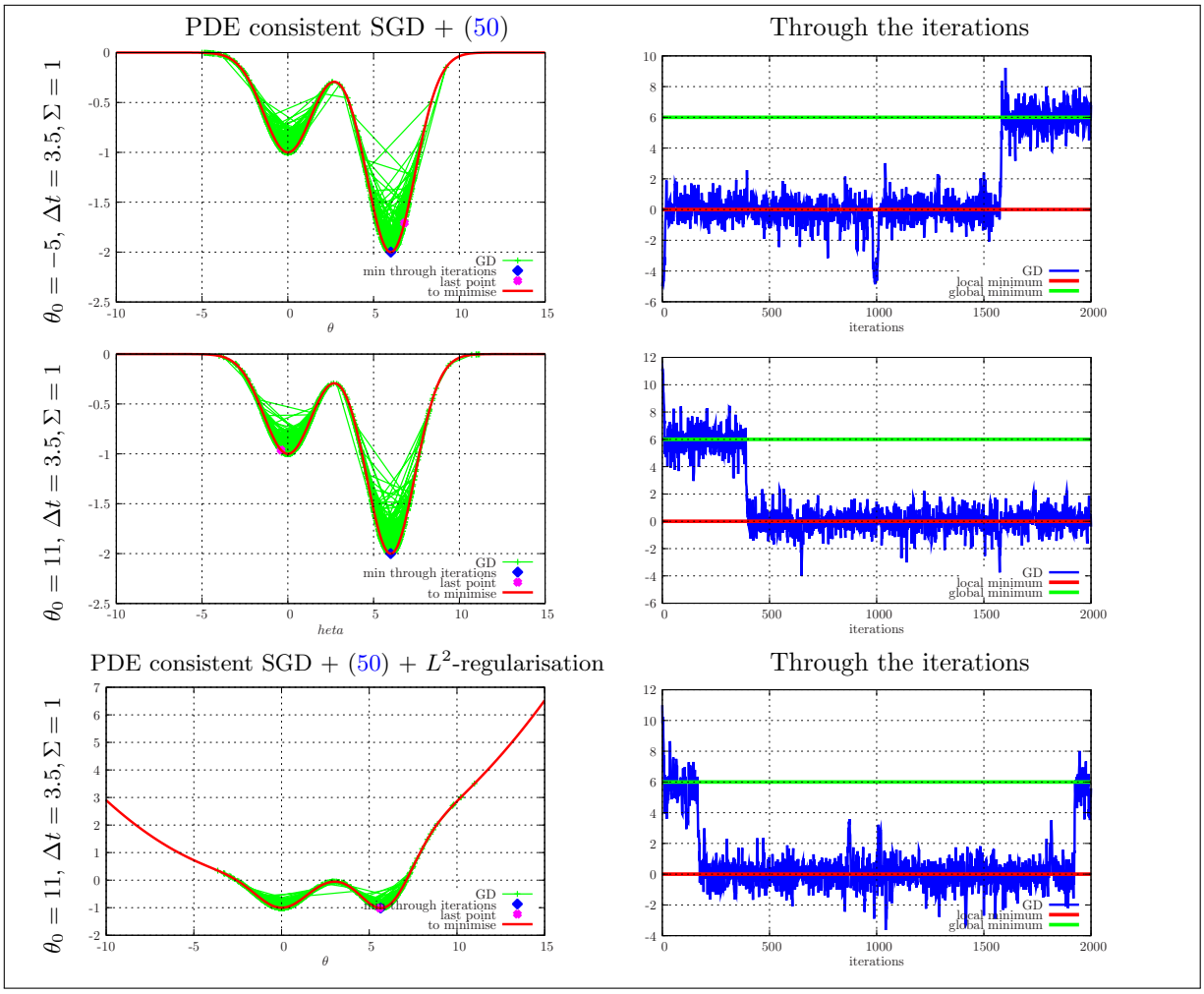

Figure 6: These figures revisit the test-case of example 3.1.1 but with a PDE consistent SGD algorithm with adaptive time steps given by (50) chosen to ensure the stability of the stochastic process (43). The last line is in the same conditions as the second line of figure 3 but with the PDE consistent SGD and its CFL condition. Except from the type of algorithm, the condition are similar to the ones of figures 2–1 in term of initial points.

Example 3.6.2 (A consistent SGD as in a PDE framework, quantitative results) *The results of example 3.6.1 related to figure 5 were mainly qualitative. We here suggest some quantitative ones. The conditions are the same as in the previous (quantitative) examples 3.1.2–3.1.4–3.5.2. The probabilities for the PDE consistent SGD algorithm to reach the vicinities of local/global minima are displayed in table 5 for several values of the learning rate/time step. The PDE consistent SGD results of table 5 can be compared to the SGD results of table 4:*

- *first, for small learning rate, the PDE consistent SGD and SGD have statistically comparable performances.*
- *On another hand, for* $\gamma = 3.50$, *the PDE consistent SGD presents an improvement and allows reaching more often the global minimum* $\theta^* = 6$ *(with probability* $\approx 40\%$ *instead of* $\approx 0.89\%$

for SGD). Figure 5 is representative of the behaviour of the algorithm in this configuration.

- *The narrower vicinities are not better explored with the PDE consistent SGD than with classical SGD, cf. last line of table 5.*
- *For $\gamma = 25.0$, the performances of SGD or of the PDE consistent SGD are statistically equivalent. Table 6 presents results in the same conditions but with an L^2-regularisation with $\xi = 0.029$ and $\theta_{guess} = 0$: the penalised PDE consistent SGD presents poor performances. This is mainly due to the frequent appearance of numerical instabilities (as for SGD in figure 4 last line).*

Now, as explained in example 3.6.1, the PDE consistent SGD comes with interesting numerical tools such as the time step limitation (50). Table 7 presents the results obtained with the PDE consistent SGD with time step limitation (50): independently of the initial choice of the learning rate γ/time step Δt, the results are statistically equivalent. This is because the time step is updated on-the-fly *on the path of the process $(\theta_n)_{n\in\{1,\dots,n_{\text{epoch}}=2000\}}$ (note that it means that not every stochastic process ends at T in this case). Now, the PDE consistent SGD algorithm with time step limitation (50) leads to a considerable improvement: the vicinity of the global minimum is reached with a $\approx 70\%$ probability, even in the penalised case. The narrow vicinities ($\epsilon = 10^{-4}$) are significatively more explored than with SGD, see table 7.*

Initialiser $\theta_0 \sim \mathcal{U}([-15, 15])$	Vicinity of $\theta^* = 0$	Vicinity of $\theta^* = 6$
PDE consistent SGD ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10^{-2}, 10^{-2}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-2}, 6 + 10^{-2}])$
$\gamma = 0.05$, $\Sigma = 1$	$23.8\% \in [22.96\%, 24.63\%]$	$29.0\% \in [28.15\%, 29.93\%]$
$\gamma = 3.50$, $\Sigma = 1$	$0.10\% \in [0.038\%, 0.172\%]$	$40.0\% \in [39.08\%, 41.02\%]$
$\gamma = 25.0$, $\Sigma = 1$	$0.13\% \in [0.059\%, 0.201\%]$	$0.29\% \in [0.184\%, 0.396\%]$
PDE consistent SGD ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10^{-4}, 10^{-4}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-4}, 6 + 10^{-4}])$
$\gamma = 0.05$,	$0.00\% \in [0.00\%, 0.00\%]$	$0.06\% \in [0.012\%, 0.108\%]$

Table 5: Probabilities for the PDE consistent SGD algorithm to recover the local ($\theta^* = 0$) or the global ($\theta^* = 6$) minimum of (4) for several learning rates γ. To compute the probabilities, we have resort to $N_{seed} = 10^4$ multiple initialisations with $\theta_0 \sim \mathcal{U}([-15, 15])$. Confidence intervals for the results are provided. Comments are provided in example 3.6.2.

Initialiser $\theta_0 \sim \mathcal{U}([-15, 15])$	Vicinity of $\theta^* = 0$	Vicinity of $\theta^* = 6$
Penalised PDE consistent SGD ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10^{-2}, 10^{-2}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-2}, 6 + 10^{-2}])$
$\gamma = 25.0$, $\xi = 0.029$, $\Sigma = 1$	$0.00\% \in [0.000\%, 0.000\%]$	$0.00\% \in [0.000\%, 0.000\%]$

Table 6: Probabilities for the penalised PDE consistent SGD algorithm to recover the local ($\theta^* = 0$) or the global ($\theta^* = 6$) minimum of (4) for $\gamma = 25.0$ and $\xi = 0.029$. To compute the probabilities, we have resort to $N_{seed} = 10^4$ multiple initialisations with $\theta_0 \sim \mathcal{U}([-15, 15])$. Confidence intervals for the results are provided. The 0.00% are not significative but attest for a small probability. Comments are provided in example 3.6.2.

This section was mainly illustrative. We here would like to sum-up what we think might lead to gains for ML algorithms:

- first, with the previous examples 3.6.1–3.6.2, we showed that

Initialiser $\theta_0 \sim \mathcal{U}([-15, 15])$	Vicinity of $\theta^* = 0$	Vicinity of $\theta^* = 6$
PDE consistent SGD + (50) ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10^{-2}, 10^{-2}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-2}, 6 + 10^{-2}])$
$\gamma = 0.05$, $\Sigma = 1$	$3.30\% \in [2.94\%, 3.65\%]$	$69.57\% \in [68.66\%, 70.47\%]$
$\gamma = 3.50$, $\Sigma = 1$	$3.24\% \in [2.89\%, 3.58\%]$	$70.44\% \in [69.54\%, 71.33\%]$
$\gamma = 25.0$, $\Sigma = 1$	$3.09\% \in [2.75\%, 3.42\%]$	$69.93\% \in [69.03\%, 70.82\%]$
PDE consistent SGD + (50) ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10^{-4}, 10^{-4}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-4}, 6 + 10^{-4}])$
$\gamma = 0.05$, $\Sigma = 1$	$0.02\% \in [0.00\%, 0.047\%]$	$4.820\% \in [4.400\%, 5.240\%]$
Penalised PDE consistent SGD + (50)	$\mathbb{P}(\theta_{\min} \in [-10^{-2}, 10^{-2}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-2}, 6 + 10^{-2}])$
$\gamma = 0.05$, $\Sigma = 1$, $\xi = 0.029$	$3.17\% \in [2.82\%, 3.51\%]$	$69.89\% \in [68.99\%, 70.78\%]$

Table 7: Probabilities for the (penalised or not) PDE consistent SGD algorithm with adaptive time steps provided by (50) to recover the local ($\theta^* = 0$) or the global ($\theta^* = 6$) minimum of (4) for several learning rates γ. To compute the probabilities, we have resort to $N_{seed} = 10^4$ multiple initialisations with $\theta_0 \sim \mathcal{U}([-15, 15])$. Confidence intervals for the results are provided. Comments are provided in example 3.6.2.

  - classical ML algorithms can be revisited in a PDE framework and can be resumed to modeling choices (of $\alpha, \theta^* - \theta, \mathrm{d}\mathcal{P}_X^{N,n}, \hat{u}$). More examples will come in the next sections, attesting for its relevance and generality.
  - New algorithms can be designed from the PDE framework we detailed above. The consistent PDE framework can for example ensure a better control of the stochasticity without overcost (with respect to SGD for example which only allows discrete diffusion coefficients depending on the batchsize n and inducing an overcost $\times \frac{N}{n}$).
  - It introduces a notion which is classical for PDE: consistency. It helps understanding in which situations the solution of one set of PDEs can coincide with the solution of another set of PDEs (see the discussions of remark 3.5–3.6). Consistency can be ensured by astutely choosing $\alpha, \theta^* - \theta, \mathrm{d}\mathcal{P}_X^{N,n}, \hat{u}$ so that (27) gets the closer possible to (7) without relying on a particular regime ($\delta \sim 0$).

  As a consequence, the framework is more general hence richer and may help understanding what can be reached in term of solution for an ML algorithm.

- Second, the PDE framework comes with many theoretical results:
  - the time step control for stability and accuracy is an example, probably the simplest one. In examples 3.6.1–3.6.2, we even show that it is precisely this ingredient which induces an important improvement. A relevant time step limitation could also have been applied to GD and SGD now that we identified the PDE the stochastic processes asymptotically solve. Example 3.6.3 is another immediate application of our PDE framework.
  - We insist on the fact that in order to be able to exhibit a relevant time step control, we need to be able to characterise the expressions of μ, σ depending on $\alpha, \theta^* - \theta, \mathrm{d}\mathcal{P}_X^{N,n}$. In order to obtain their expressions, it is important being able to come back to the PDE solved by the simulated stochastic process.
- Third, the test-cases in the different examples are simple, coded in small .py files. Those are precious for an eventual verification of an implementation in more general ML frameworks.

With the above points, we briefly highlighted some possibilities from the PDE framework we introduced for ML. Another example is presented below in example 3.6.3 and a limitation of the drift-diffusion PDE framework is presented in example 3.6.4.

Example 3.6.3 (The Fokker Planck equation of (47)) *In this example, we consider the Fokker-Planck equation of (47) given by*

$$\partial_t p(t,\theta) + \nabla_\theta \left[J'(\theta)p(t,\theta)\right] - \nabla^2_{\theta,\theta}\left[\frac{1}{2}|J'(\theta)|\Sigma^2 p(t,\theta)\right] = 0. \tag{51}$$

For long times, the solution of the above equation is given by the Laplace distribution

$$p_\infty(\theta) \propto e^{sgn(J'(\theta))\frac{4}{\Sigma^2}\theta}. \tag{52}$$

*The above distribution is representative of where the process $\overline{\theta}_t$ of example 3.6.1 spends some time. The histogram of the stochastic path of $(\theta_n)_{n\in\{1,\dots,50000\}}$ is displayed in figure 7: it has important probabilities of sampling in the vicinities of the modes $\theta^* = 0$ and $\theta^* = 6$. With the choice of $\alpha, \theta^* - \theta$ of example 3.6.1, the stochastic process, on average, spends as much time around $\theta^* = 0$ as around $\theta^* = 6$. Other choices may guaranty different behaviours. By controlling (*via *the choices of $\alpha, \theta^* - \theta$ etc.) the asymptotical solution of the Fokker-Planck equation, we may build algorithms which, by construction, can have desired properties.*
Finally, once $\alpha, \theta^* - \theta$, $\mathrm{d}\mathcal{P}_X^{N,n}$ chosen, building the Fokker Planck equation and approximating p_∞ may be at hand. The asymptotical distribution p_∞ could then be used as an initialiser.

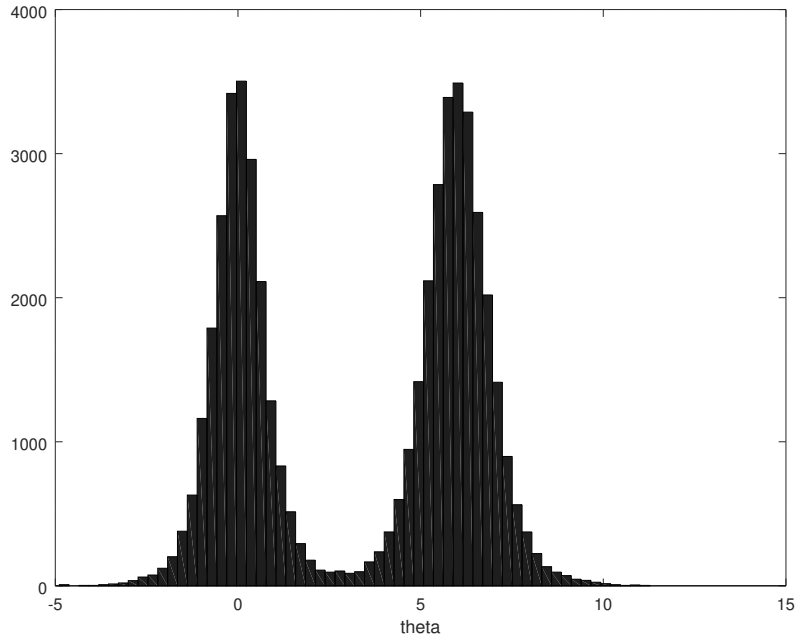

Figure 7: Distribution of the sampled path $(\theta_n)_{n\in\{1,\dots,n_{\text{epoch}}=20000\}}$ in the same conditions as example 3.6.3. The discretised distribution recovers the one predicted by the stationary solution (52) of the Fokker-Planck equation (51).

Example 3.6.4 (Two local minimum separated by a large flat spot) *In this example, we revisit the results of the new consistent SGD algorithm obtained from a PDE framework on a modification of functional (4) given by*

$$\theta \in \mathbb{R} \longrightarrow J(\theta) = -e^{-\frac{(\theta+10)^2}{4}} - 2e^{-\frac{(\theta-6)^2}{4}}. \tag{53}$$

The function has the same performance as (4) but its two minima are located at $\theta^* = -10$ and $\theta^* = 6$ with $J(6) < J(-10)$ and are separated by a flat spot with a gradient which is almost zero. *The results obtained with the PDE consistent SGD with adaptive time steps given by (50) are displayed in figure 8 in the same conditions (same initial guess etc.) as in the previous examples. For both initialisations, $\theta_0 = -5$ and $\theta_0 = 11$, the algorithm finds accurately the locations of the minima. But the algorithm seems to lose its exploration property in the sense that the process does not jump from one local minima to the other. In fact, even a fine tuning of the parameters $\Delta t, \Sigma$ does not allow jumping for this example. In this situation, which is extremely difficult to detect in practice when we do not have access to the functional J (i.e. in high dimensions for example), the only way to be sure to jump is to change the guess θ_0, i.e. to perform a new initialisation, i.e. to rely on the initialiser. We recover the con of deterministic methods such as the Newton algorithm, see example 3.1.1, but with non optimal steps. From a pessimistic point of view, one could argue that we have the cons of a stochastic solver and the cons of a deterministic one. From an optimistic point of view, we recover the behaviour of a deterministic algorithm but with a stochastic one. Indeed, with the time step control, we can ensure a certain accuracy: this means that if we choose the accuracy (a threshold under which we would consider the gradient is satisfactory, let us say $\varepsilon = 10^{-7}$) as a parameter, we can trigger a reinitialisation from a different θ_0. We will see in section 4 that such idea was intrisically embedded in the learning framework based on transport hinted at in [54].*

Example 3.6.5 (Two minima separated by a large flat spot, quantitative results) *The results of example 3.6.4 related to figure 8 were mainly qualitative. We here suggest some quantitative ones. The conditions are the same as in the previous (quantitative) examples 3.1.2–3.1.4–3.5.2–3.6.2.*

The probabilities for the (penalised or not) PDE consistent SGD algorithm to reach the vicinities of the local/global minima $\theta^* = -10$ and $\theta^* = 6$ are displayed in table 8. In both cases (penalised or not), we have an important probability (64.52% and 55.59%) of recovering the vicinity of the global minimum $\theta^* = 6$. The probability of recovering the vicinity of the local minimum $\theta^* = -10$ is also important (35.47% and 41.87%) attesting for a small probability for the algorithm to remain stuck in zero-gradient regions (2.53% and 0.01%). But the stochastic process almost never jumps from one vicinity to the other (cf. behaviour displayed in figure 8). Note that in this same configuration, a modified Newton algorithm (as in example 3.1.1) gives similar probabilities.

Finally, SGD or the PDE consistent SGD briefly presented in this section, by introducing stochasticity in order to introduce a second order diffusion term with respect to GD, enters a class of methodology denoted by 'noise injection methods', see [49]. Dropout, tackled in the next section 3.6.2, also enters this class.

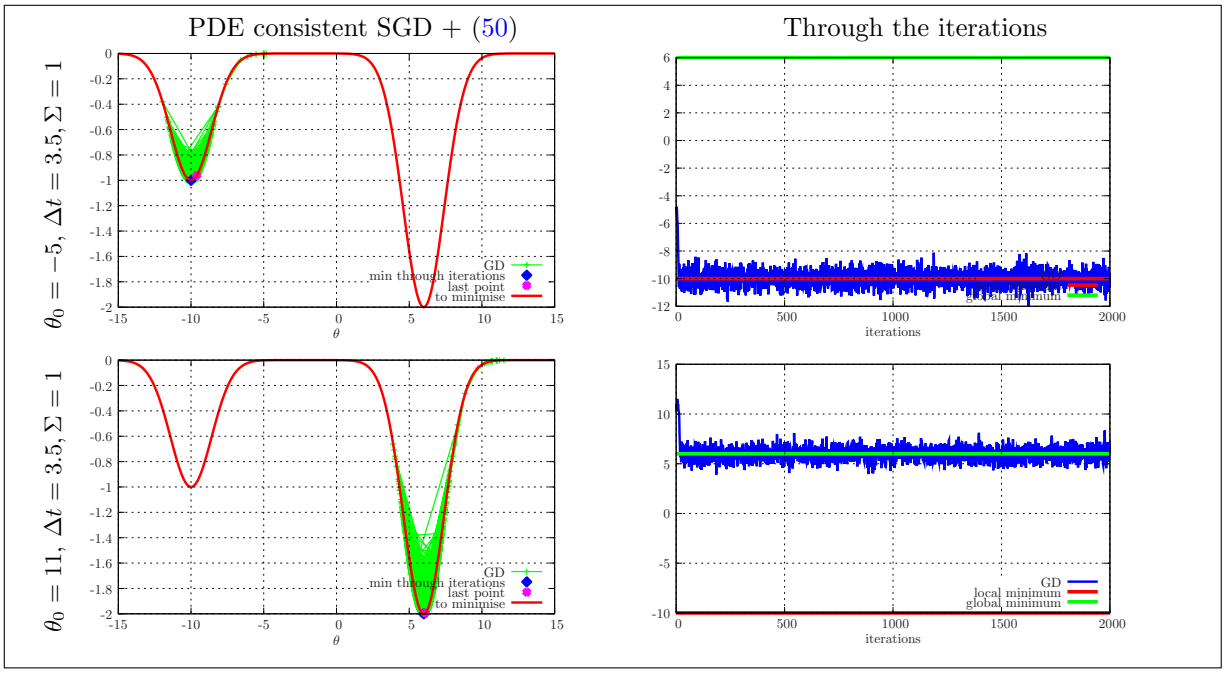

Figure 8: These figures revisit the test-case of example 3.1.1 but *with local minima further away from each others and a large almost zero-gradient vicinity between them* with a PDE consistent SGD algorithm with adaptive time steps given by (50) chosen to ensure the stability of the stochastic process (43). The last line is in the same conditions as the second line of figure 3 but with the PDE consistent SGD and its CFL condition. Except from the type of algorithm, the condition are similar to the ones of figures 2–1 in term of initial points.

Initialiser $\theta_0 \sim \mathcal{U}([-15, 15])$	Vicinity of $\theta^* = -10$	Vicinity of $\theta^* = 6$
PDE consistent SGD + (50) ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10 - 10^{-2}, -10 + 10^{-2}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-2}, 6 + 10^{-2}])$
$\gamma = 3.50,\ \Sigma = 1$	$41.87\% \in [40.90\%, 42.83\%]$	$55.59\% \in [54.61\%, 56.56\%]$
PDE consistent SGD + (50) ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10 - 10^{-4}, -10 + 10^{-4}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-4}, 6 + 10^{-4}])$
$\gamma = 3.50,\ \Sigma = 1$	$8.59\% \in [8.040\%, 9.139\%]$	$9.53\% \in [8.954\%, 10.105\%]$
Penalised PDE consistent SGD + (50)	$\mathbb{P}(\theta_{\min} \in [-10^{-2}, 10^{-2}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-2}, 6 + 10^{-2}])$
$\gamma = 3.50,\ \Sigma = 1,\ \xi = 10^{-4}$	$35.47\% \in [34.53\%, 36.40\%]$	$64.52\% \in [63.58\%, 65.45\%]$

Table 8: Probabilities for the (penalised or not) PDE consistent SGD algorithm with adaptive time steps provided by (50) to recover the local ($\theta^* = -10$) or the global ($\theta^* = 6$) minimum of (53) for several learning rates γ. To compute the probabilities, we have resort to $N_{seed} = 10^4$ multiple initialisations with $\theta_0 \sim \mathcal{U}([-15, 15])$. Confidence intervals for the results are provided. Comments are provided in example 4.2.2.

3.6.2 Examples of other particular choices for $\alpha, \theta^* - \theta$: dropout

In this section, we want to insist on the fact that dropout (and certainly many other ML ingredients) can be recast into the same PDE framework as GD and SGD. During the training, dropout is nothing more than an additional ingredient which artificially adds fluctuations, stochasticity, but without considerably affecting the computational time such as minibatching. It is nothing more than another

'noise injection technic' [49], at least during the training. In other word, in our framework, dropout (for the training) only consists in a particular choice of $\mathrm{d}\mathcal{P}_X^{N,n}$ (see the SGD analysis above).

What is astonishing and new with respect to what we previously analysed is rather how dropout is used during the testing phase (i.e. when evaluating $\overline{J}$), not during the training: let us introduce, for an epoch/iteration j, a vector $B_j \in \mathbb{R}^{Card(\theta)}$ of independent identically distributed random[26] samplings of a random variable having mean p. Dropout suggests replacing (33) (for GD) by

$$\begin{aligned}\theta_{j+1} &= \theta_j - \frac{\gamma}{N} B_j \otimes \sum_{k=1}^{N} L'(X_k, B_j \otimes \theta_j) \nabla_\theta u(X_k, B_j \otimes \theta_j), \\ &= \theta_j - \gamma \int B_j \otimes L'(X, B_j \otimes \theta_j) \nabla_\theta u(X, B_j \otimes \theta_j) \, \mathrm{d}\mathcal{P}_X^{N,p},\end{aligned} \tag{54}$$

where $\otimes$ denotes the element-wise product. Note that equation (54) implicitly introduces a new distribution $\mathrm{d}\mathcal{P}_X^{N,p}$ depending on random vector B_j at epoch j. Now, we would like to focus on one particularity of dropout: the weights θ learnt by the optimiser are not the ones used during the testing phase. During the *testing phase*, it is common using $p \otimes \theta_j$, the mean[27] of $B_j \otimes \theta_j$ instead of θ_j. The reason why is not obvious as, due to the nonlinearity of J, the mean of the gradient is not equal to the gradient of the averaged weights, i.e.

$$\int B_j \otimes L'(X, B_j \otimes \theta_j) \nabla_\theta u(X, B_j \otimes \theta_j) \, \mathrm{d}\mathcal{P}_X^{N,p} \neq p \otimes L'(X, p \otimes \theta_j) \nabla_\theta u(X, p \otimes \theta_j), \text{ in general.}$$

Let us try to understand what happens here. By definition, we have[28] $B_j \sim \mathcal{L}(p)$, so that $\mathbb{E}[B_j] = p$. We also introduce the centered fluctuation of B_j, $\epsilon_j = B_j - p$. This means that $\epsilon_j \sim \mathcal{L}(0)$. Now, assume that the support of the fluctuations is small: we abusively write $\epsilon_j \sim 0$. Let us consider any transformation $x \to T(x)$, then

$$T(B_j) = T(p + \epsilon_j) \underset{\epsilon_j \sim 0}{\sim} T(p) + \epsilon_j \nabla_x T(p) + \epsilon_j^T \nabla_{x,x}^2 T(p) \epsilon_j + \mathcal{O}(\epsilon_j^3).$$

This means that we have[29]

$$\begin{aligned}\theta_{j+1} = \quad & \theta_j - \gamma \int B_j \otimes L'(X, B_j \otimes \theta_j) \nabla_\theta u(X, B_j \otimes \theta_j) \, \mathrm{d}\mathcal{P}_X^{N,p}, \\ \underset{\epsilon_j \sim 0}{=} \quad & \theta_j - \frac{\gamma p}{N} \sum_{i=1}^{N} L'(X_i, p\theta_j) \nabla_\theta u(X_i, p\theta_j) \\ & - \frac{\gamma \epsilon_j}{N} \sum_{i=1}^{N} L'(X_i, p\theta_j) \nabla_\theta u(X_i, p\theta_j) \\ & - \gamma \epsilon_j \underbrace{\frac{p}{N} \sum_{i=1}^{N} \left[L''(X_i, p\theta_j) \nabla_\theta u(X_i, p\theta_j) \nabla_\theta^T u(X_i, p\theta_j) + L'(X_i, p\theta_j) \nabla_{\theta,\theta}^2 u(X_i, p\theta_j) \right]}_{= p \nabla_{\theta,\theta}^2 J_N(p\theta_j) \underset{p \to 1}{\to} \nabla_{\theta,\theta}^2 J_N(\theta)} + \mathcal{O}(\epsilon_j^2).\end{aligned}$$

From the above equation, we can see how dropout can implicitly, without explicitly computing it, estimate the hessian of J_N when $\epsilon_j \sim 0$ and $p \to 1$. The regime $p \to 1$ echoes the regime $\delta \sim 0$ of the previous sections. Dropout can be understood as a way to estimate $\nabla_{\theta,\theta} J_N(\theta)$ without explicitly computing it. We will see in section 4 that Papanicolaou [54] already understood this possibility.

[26] In dropout, Bernouilli random variables are usually chosen but in practice, arbitrary laws can be used.

[27] As be definition we have $\int B_j \otimes \theta_j \, \mathrm{d}\mathcal{P}_X^{N,p} = p \otimes \theta_j$.

[28] Here, $B_j \sim \mathcal{L}(p)$ means that B_j follows an arbitrary distribution of law $\mathcal{L}$ having mean p and finite variance.

[29] By taking $B_j \to T(B_j) = B_j \otimes L'(X, B_j \otimes \theta_j) \nabla_\theta u(X, B_j \otimes \theta_j)$ such that $T(p + \epsilon_j) \underset{\epsilon_j \sim 0}{\sim} pL'(X, p\theta_j) \nabla_\theta u(X, p\theta_j) + \epsilon_j \left[L'(X, p\theta_j) \nabla_\theta u(X, p\theta_j) + pL''(X, p\theta_j) \nabla_\theta u(X, p\theta_j) \nabla_\theta^T u(X, p\theta_j) + pL'(X, p\theta_j) \nabla_{\theta,\theta}^2 u(X, p\theta_j) \right] + \mathcal{O}(\epsilon_j^2)$.

3.7 Reminder of the pros and cons of the different presented approaches

So far, we have attempted to rewrite classical ML algorithms (Newton, GD, SGD, SGD+Dropout) into a PDE framework. We also intensively exploited this PDE framework in order to suggest new learning algorithms (a new SGD, control of the learning rate for stability/accuracy...), a new way to define a consistent optimiser which leads to new possible consistent heuristics (by choosing $\alpha, \theta^* - \theta$, $\mathrm{d}\mathcal{P}_X^{N,n}, \hat{u}$). We also have highlighted a certain number of pros and cons of those (non-exhaustive) list of examples. Let us recall them here:

1. Deterministic Newton algorithm:

   pro Optimal number of iterations in order to reach a local minimum.

   pro Stable.

   pro Does ensure cancelling the gradient.

   pro Consistent with the problem we want to solve.

   con Intractable in high dimension: impossible to compute the (absolute value of the) hessian.

   con Remains stuck if initialized where the gradient is zero. Penalisation can be used in order to avoid zones with zero-gradient but at the cost of a loss of performance: the bigger the penalisation coefficient, the faster the algorithm drifts toward the region of interest but the worse the performances.

   con Does not natively take into account negative eigenvalues of the hessian: existence of saddle points and need for the computations (or approximations) of the absolute value of the hessian.

   con Does not natively take into account the important number of local minima (exponentially many local minima with d_{in} and N, see [4]): the algorithm needs several initialisations at several different starting points and this deterministic algorithm can not face its exponential growth with d_{in} and N, see [4].

2. Gradient Descent (GD)

   con sub optimal number of iterations in order to reach a local minimum: the (absolute value of the) hessian is only coarsely approximated.

   pro tractable in high dimension: no need to compute the (absolute value of the) hessian.

   con May be inconsistent with the problem we want to solve (see the $\delta \sim 0$ discussions).

   con Remains stuck if initialized where the gradient is zero. Penalisation can be used in order to avoid zones with zero-gradient but at the cost of a loss of performance (same as above for Newton).

   pro Natively tries to take into account negative eigenvalues of the hessian: it makes sure the algorithm goes in the opposite direction of the gradient.

   con Does not natively take into account the important number of local minima (exponentially many local minima with d_{in} and N, see [4]): the algorithm needs several initialisations at several different starting points.

   con Does not ensure cancelling the gradient.

   con may be instable: with bounded oscillations or unbounded ones which go to infinity.

   con Remains stuck if the process jumps into a 'flat spot' (close to zero gradient vicinity).

3. Stochastic Gradient Descent and its modifications (we also put noise injection methods inthere)

   con sub optimal number of iterations in order to reach a local minimum.

- pro tractable in high dimension (no need to compute the hessian).
- con May be inconsistent with the problem we want to solve (see the $\delta \sim 0$ discussions) but still, more consistent than GD.
- pro may not remain stuck if initialized where the gradient is zero.
- pro Natively takes into account negative eigenvalues of the hessian: it makes sure the algorithm goes in the opposite direction of the gradient.
- pro Natively takes into account the important number of local minima (exponentially many, see [4]): the stochasticity may help jumping from one region with a local minima to another one.
- con Sometimes, the stochasticity is not enough to jump and 'more' noise has to be injected.
- con Does not ensure cancelling the gradient.
- con may be instable: with bounded oscillations or unbounded ones which go to infinity.
- con Some algorithms inducing stochasticity (such as minibatching for example) lead to higher computational times.
- con Need to finely tune the amount of stochasticity (if too low $\Longrightarrow$ same as GD, if too important $\Longrightarrow$ bad performances, if well-tuned $\Longrightarrow$ good exploration of the space of the weights θ).
- con Remains stuck if the process jumps into a 'flat spot' (close to zero gradient vicinity).
- pro Allows implicitly estimating the hessian of J_N (cf. the dropout discussion) if the parameters of the noise injection method are well suited.

4. The new PDE consistent SGD

- pro interesting number of iterations in order to reach a local minimum.
- pro tractable in high dimension (no need to compute the hessian).
- con May be inconsistent with the problem we want to solve (see the $\hat{u} \sim \delta \sim 0$ discussions) but still, more consistent than GD and SGD.
- pro may not remain stuck if initialized where the gradient is zero.
- pro Natively takes into account negative eigenvalues of the hessian.
- pro Natively takes into account the important number of local minima (exponentially many, see [4]): the stochasticity may help jumping from one region with a local minima to another one.
- pro stability/accuracy conditions on the learning rate can be derived and are efficient ...
- con ... but the stability/accuracy conditions and accuracy conditions are computationally intensive (in general, i.e. in high dimension, it will need a reduction, i.e. a parallel communication).
- con Does not ensure cancelling the gradient.
- con Need to finely tune the amount of stochasticity (if too low $\Longrightarrow$ same as GD, if too important $\Longrightarrow$ bad performances, if well-tuned $\Longrightarrow$ good exploration of the space of the weights θ)...
- pro ... but at least, the stochasticity can be chosen simply, arbitrarily, continuously and do not lead to higher computational times.
- con Remains stuck if the process jumps into a 'flat spot' (close to zero gradient vicinity).
- pro Implicitly estimates the hessian of J_N (cf. the dropout discussion) if the parameters of the noise injection method are well suited.

pro Offers a simple and clear formalism: every existing algorithms can be recast in this framework. They can be analysed thanks to this framework (consistency mainly). Building new algorithms resume to particular choices of $\alpha, \theta^* - \theta$, $\mathrm{d}\mathcal{P}_X^N, \hat{u}$ and their respective effects are clearly[30] identified.

Up to this point, we can not really say that we improved a lot existing algorithms. We only improved our understanding of the learning algorithms, we also have a better idea of what we want from them, and we put them in a well-known and convenient framework. For example, some desired behaviours for our optimiser are listed below:

- it should not go uphill or remain stuck on saddle points (mandatory),
- it should try to cancel the gradient (mandatory) or at least ensure having the gradient under a certain threshold,
- it should be able to jump from one local minimum to another and do not spend to much time in a vicinity of interest, i.e. it must have good exploration properties (mandatory),
- it should be able to use big time steps far from local minimum and automatically refine it in the interesting vicinities,
- it should be stable (mandatory),
- a control of the error would be a plus,
- it should not have too many hyperparameters to tune which could have contradictory effects and could be hard to understand (mandatory),
- it should estimate the hessian of J_N without explicitly computing it.

None of the previous algorithms gathers all the above properties, not even the mandatory ones. But before tackling some new algorithms, there remains one important question: the question of $\overline{u}$ and $\hat{u}$. So far, every presented algorithms aimed at solving a PDE of solution $\overline{u}(t, \theta)$ whereas $\overline{u}$ is only the mean component of $u(t, \theta, X)$[31]. In which case fitting parameters θ on $\overline{u}$ is enough? Probably if the two last lines of (27) are negligible. Is it possible to characterise when those lines are negligible? If those lines are negligible then we can expect the best result θ^* of a learning session on $\overline{u}(\theta)$ to give satisfactory results on $u(\theta, X)$. But otherwise? Can we know if the two last lines of (27) are negligible or not?

3.8 Shall we consider $\hat{u} \neq 0$? And how?

At this stage of the discussion, the question of what we can do to take $\hat{u} \neq 0$ into account remains an open problem. The problem is not even well-posed in the sense that with equation (27), even once $\theta^* - \theta$ chosen, we only have 1 equation on $\overline{u}$ but 2 unknowns $\overline{u}$ and $\hat{u}$. Of course, we can

- try to cancel the whole solution by arbitrarily choosing a closure equation for $\hat{u}$ based on some additional heuristics (as done in turbulence modeling for example [46]). But this closure must not interfere with the case when we do not need it (i.e. when the two last lines of (27) are negligible). It may lead to a new consistency problem which we tried to solve so far with the introduction of our new SGD algorithm. In other words, being able to detect when the two last lines of (27) are negligible or not would be very convenient.

[30] Or at least, it is clearer for the numerician, familiar with such PDE framework.
[31] Recall we have $u(t, \theta, X) = \overline{u}(t, \theta) + \hat{u}(t, \theta, X)$ with $\overline{u}(t, \theta) = \int u(t, \theta, X)\, \mathrm{d}\mathcal{P}_X$.

– We can also try to consider weights parametered by X, i.e. $\theta(X)$, see (22). The work of [24] can be reinterpreted as an attempt to work on $\hat{u}$, and in this sense, is singular with respect to what can be found in the ML literature. In order to take into account the dependence with respect to X of the weights θ, the author tries to identify their probabilistic distributions from the dataset. A better understanding of the work of [24] in a PDE framework would certainly be really interesting (several complex questions remain and must be more thoroughly studied: for example, if $\theta(X)$, then the architecture is of the form $u(\theta(X), X)$, etc.).
Still, thanks to the PDE framework we previously derived, we can rely on quite a number of publications suggesting ways to deal with such additional dimensions X (which can be reinterpreted as uncertain parameters). The MC schemes used in an uncertainty propagation context, cf. [59, 57, 55, 73, 12, 62, 61], could certainly be of interest in this ML context due to the similar structures/formalisations of the problems.

In brief, there are probably many other heuristic solutions but before relying on them, we would like to investigate on a remark made in [54] and the relation between *transport in the diffusion limit and learning theory*. We will see that it allows, in a way, understanding under which conditions $\overline{u}$ is enough and under which conditions something must be done on $\hat{u}$ in order to improve the performances of our ML architecture.

4 Transport and the diffusion limit

In this section, we try to bridge the gap between transport and diffusion for machine learning, as put forward in [54]. So far, we have listed a serie of pros and cons of ML algorithms and recast some of them in a PDE framework. We now would like to investigate on how solving the transport equation instead of the drift-diffusion one (as suggested in [54]) may help us in our ML context.
The next section 4.1 is only a reminder of how the transport equation can degenerate toward the drift-diffusion one in a specified regime/limit. The analysis and the calculations performed in this section are classical in photonics and neutronics, see [72, 16, 48, 17, 42, 13]. The reader familiar with this topic can easily skip the section. But the paragraph is still interesting on many points with respect to ML algorithms (see the remarks within).
Finally, in section 4.2, we perform some analogies between the transport framework suggested in [54] and PDE based ML algorithms. We even design a transport based learning algorithm which gathers many of the desired properties tackled in section 3.7. We apply it to the *fil rouge* problem of this document and to a 3-layer Deep Neural Network approximating Runge's function (see example 3.1.1).

4.1 Transport and diffusion limit: the steps on a classical example

This section aims at bridging the gap between the notations of [54, 26] and ours. The idea is to identify the different steps allowing to recover the diffusion limit from the transport equation. We will then apply the same material to a better-chosen transport equation in order to solve an ML problem in section 4.2.
Let us here consider the transport equation given by

$$\partial_t u(x,t,v) + F^T(x,t,v)\partial_x u(x,t,v) + \sigma(x,t)u(x,t,v) = \sigma(x,t)\int P(x,t,v,v')u(x,t,v')\,\mathrm{d}v'. \tag{55}$$

In the above equation, $x \in \mathbb{R}^n, t \in \mathbb{R}^+, v \in \mathbb{R}^{d_{in}}$ and $u \in \mathbb{R}$ (i.e. $d_{out} = 1$). We also assume that $F \in \mathbb{R}^n$, $\int F(x,t,v')\,\mathrm{d}v' = 0, \forall x,t$. We furthermore have $\sigma(x,t,v,v') = \sigma(x,t)P(x,t,v,v')$ and $\int P(x,t,v,v')\,\mathrm{d}v' = 1, \forall x,t,v$ et $\int P(x,t,v,v')\,\mathrm{d}v = 1, \forall x,t,v'$. If we rewrite the above transport

equation with respect to the non-dimensional quantities

$$\begin{aligned} t^* &= \tfrac{t}{\mathcal{T}}, \\ x^* &= \tfrac{x}{\mathcal{X}}, \\ F^* &= \tfrac{F}{\mathcal{F}}, \\ \sigma^* &= \tfrac{\sigma}{\Lambda}, \end{aligned} \tag{56}$$

we get (the * upperscripts have been dropped for conciseness)

$$\partial_t u(x,t,v) + F^T(x,t,v)\frac{\mathcal{F}\mathcal{T}}{\mathcal{X}}\partial_x u(x,t,v) + \mathcal{T}\Lambda\sigma(x,t)u(x,t,v) = \mathcal{T}\Lambda\sigma(x,t)\int P(x,t,v,v')u(x,t,v')\,\mathrm{d}v'.$$

Suppose now that

$$\frac{\mathcal{F}\mathcal{T}}{\mathcal{X}} \sim \frac{1}{\delta} \text{ and } \mathcal{T}\Lambda \sim \frac{1}{\delta^2}. \tag{57}$$

In appendix A, we perform a Hilbert developpement [27, 64, 29] of u with respect to[32] $\delta \sim 0$, i.e. we develop $u = u_0 + \delta u_1 + \delta^2 u_2 + \mathcal{O}(\delta^3)$, and build the equation satisfied by u_0. We obtain that u_0 satisfies

$$\partial_t u_0(x,t) - \partial_x\left[\left(\int F(x,t,v)b(x,t,v)\,\mathrm{d}v\right)\partial_x u_0(x,t)\right] = 0, \tag{58}$$

where b is closely related to σ, P and F (see appendix A for more details). For photonics for example, $\int F(x,t,v)b(x,t,v)\,\mathrm{d}v = \int vb(x,t,v)\,\mathrm{d}v$ can be explicited and is given by $\frac{|v|}{3\sigma(x,t)}$, see [60, 72, 16, 48, 17, 42, 13].
The proof for obtaining the diffusion limit of the transport equation is recalled in appendix A but we insist it is very classical and can be found in many publications, see [54, 39, 72, 16, 48, 17, 42, 13, 63, 60]. Let us assume some smoothness on $x \to b(x,t,v)$. Then we can expand the above expression to get

$$\partial_t u_0(x,t) - \underbrace{\left[\partial_x\left(\int F(x,t,v)b(x,t,v)\,\mathrm{d}v\right)\right]}_{\mu(x,t,v)=\partial_x\gamma(x,t,v)}\partial_x u_0(x,t) - \underbrace{\left(\int F(x,t,v)b(x,t,v)\,\mathrm{d}v\right)}_{\gamma(x,t,v)}\partial^2_{x,x}u_0(x,t) = 0.$$

What is interesting here is that by solving the transport equation (55) in a particular regime, we are able to compute the solution of a drift-diffusion equation involving $\partial_x\gamma$, the derivative of γ: in practice, we do not need to compute this derivative with MC schemes (see [54, 63]) during the resolution of the transport equation. *MC schemes built in order to solve the transport equation, even in the diffusion limit, only need the evaluation of γ, see [54].* This is typically what we need in our ML algorithm: an MC scheme which implicitly computes $|\nabla^2_{\theta,\theta}J_N(\theta)|$, the absolute value of the hessian of J, but only needs evaluating numerically $\nabla_\theta J_N(\theta)$. Furthermore, the transport framework also offers interesting properties which could be useful in an ML context, see for example the next remark.

Remark 4.1 *Let us assume that*

- *$P(x,t,v,v')$ is invariant by orthogonal transformations (rotations). This means that*

$$P(x,t,Qv,Qv') = P(x,t,v,v'), \forall \text{ rotation } Q, \forall x,t.$$

  This hypothesis is verified for example if $P(x,t,v,v') = P(x,t,v\cdot v')$.

- *Let us also assume that $\mathrm{d}v$ verifies the invariance property*

$$\int g(Qv)\,\mathrm{d}v = \int g(v)\,\mathrm{d}v, \forall g \in \mathcal{C}^0, \forall \text{ rotation } Q.$$

  This hypothesis is verified for example on the uniform sphere of dimension $n-1$. This is also verified for $m(v)\,\mathrm{d}v$ if $m(v) = \tilde{m}(|v|)$.

[32] Here, δ should recall the quantity δ introduced in section 3.5: the analysis is here more detailed.

– *Under the above two hypothesis,* $b_i(v) = \tilde{b}(|v|)v_i, \forall i \in \{1, ..., n\}$ *and*

$$\int v_i b_j(x,t,v)\,\mathrm{d}v = \frac{1}{n}\int |v|^2 \tilde{b}(|v|)\,\mathrm{d}v\delta_{i,j}.$$

This means the diffusion term is characterised by a scalar.

In ML algorithms, invariance by rotation or translation (or by any arbitrary bijective transformation) can be of interest. The scattering kernel of the transport equation can, at least theoretically, be built in order to satisfy some invariance properties. It is intensively used in neutronics for example (scattering angle and velocity in the center of mass reference frame [25, 40, 10, 9]). The question here is: is such a framework relevant in order to make ML algorithms bear interesting invariant properties? Ongoing researches will be carried out in this direction.

The previous lines aimed at recovering the classical diffusion limit for neutronics or photonics. We are going to apply the material to our *fil rouge* problem and give an idea of what can be expected from the transport equation in order to build learning processes (idea briefly developed in [54]).

4.2 A transport equation degenerating toward an ML consistent diffusion equation

From the previous developpements, we suggest revisiting transport equation (55) and its diffusion limit (58) from an ML point of view. Let us first simply take (58) and replace x by θ, v, $\mathrm{d}v$ by X, $\mathrm{d}\mathcal{P}_X$ and u_0 by $\overline{u}$. We get

$$\partial_t \overline{u}(t,\theta) - \partial_\theta\left[\left(\int F(t,X,\theta)b(t,X,\theta)\,\mathrm{d}\mathcal{P}_X\right)\partial_\theta \overline{u}(t,\theta)\right] = 0. \tag{59}$$

If we do the same in the transport equation from which (59) is the limit, we get

$$\begin{aligned}\partial_t u(t,X,\theta) + F^T(t,X,\theta)\partial_\theta u(t,X,\theta)& \\ +\sigma(t,\theta)u(t,X,\theta)) = \sigma(t,\theta)\int P(t,\theta,X,X')u(t,X',\theta)\,\mathrm{d}\mathcal{P}'_X.&\end{aligned} \tag{60}$$

The equation (59) echoes (27) with the two last lines (the ones involving $\hat{u}$) cancelled whereas the solution $u(t,X,\theta)$ of equation (60) depends also on X. In other words, the regime (57) defined by $\delta \to 0$ characterises under which conditions $u(t,X,\theta)$ behaves as $\overline{u}(t,\theta)$ (i.e. under which conditions we have $\hat{u} \sim 0$). If we expand the summation (i.e. if we avoid the matricial form), (59) becomes

$$\begin{aligned}0 = \;& \partial_t \overline{u}(t,\theta) \\ & - \sum_{i=1}^{Card(\theta)} \partial_{\theta_i}\overline{u}(t,\theta) \sum_{j=1}^{Card(\theta)} \partial_{\theta_j}\left[\int F_i(t,\theta,X)b_j(t,X,\theta)\,\mathrm{d}\mathcal{P}_X\right] \\ & - \sum_{i,j=1}^{Card(\theta)}\left[\int F_i(t,X,\theta)b_j(t,X,\theta)\,\mathrm{d}\mathcal{P}_X\right]\partial^2_{\theta_i,\theta_j}\overline{u}(t,\theta),\end{aligned}$$

which is to be compared to the three first lines of (27) recalled below (but rewritten slightly differently in order to make the analogy explicit)

$$\begin{aligned}0 = \;& \partial_t \overline{u}(t,\theta) \\ & + \sum_{i=1}^{Card(\theta)} \partial_{\theta_i}\overline{u}(t,\theta) \sum_{j=1}^{Card(\theta)} \partial_{\theta_j}\left[\int \alpha_i L'(t,X,\theta)(\theta^*_j - \theta_j)\,\mathrm{d}\mathcal{P}_X\right] \\ & + \sum_{i,j=1}^{Card(\theta)}\left[\int \alpha_i L'(t,X,\theta)(\theta^*_j - \theta_j)\,\mathrm{d}\mathcal{P}_X\right]\partial^2_{\theta_i,\theta_j}\overline{u}(t,\theta).\end{aligned}$$

Now, b is related to σ, F and P: it is solution of (see appendix A)

$$\begin{cases} (I-K)b(t,X,\theta) = b(t,X,\theta) - \displaystyle\int P(t,\theta,X,X')b(t,X,\theta)\,\mathrm{d}\mathcal{P}'_X = \frac{F(t,X,\theta)}{\sigma(t,\theta)}, \\ \displaystyle\int b(t,X,\theta)\,\mathrm{d}\mathcal{P}_X = 0. \end{cases} \tag{61}$$

From the above expressions, we can identify some compatibility conditions[33] in order to identify the different terms such that:

$$F_i(t,X,\theta)b_j(t,X,\theta) = \alpha_i L'(t,X,\theta)(\theta_j^* - \theta_j).$$

Assume

- an arbitrary choice for F_i (but such that $\int F(t,X,\theta)\,\mathrm{d}\mathcal{P}_X = 0$, see appendix A),
- and an arbitrary choice for σ,
- then b_j can be obtained thanks to a pretreatment of the data, see (61),

Assume furthermore that

- $\alpha_i = -\partial_{\theta_i} u(t,X,\theta)$ (just as in GD, SGD etc.),
- then we have that the equation satisfied by $\theta_j^* - \theta_j$, mandatory to close our system of equations, is given by

$$-\frac{F_i(t,X,\theta)b_j(t,X,\theta)}{\partial_{\theta_j} u(t,X,\theta)L'(t,X,\theta)} = (\theta_j^* - \theta_j).$$

Of course, one could first choose $\theta^* - \theta$ and deduce b afterward.

In brief, the analysis of the transport equation may help us make some choices in order to solve our learning problem. Note that transport equation (60) may be even more complex (with an acceleration term for example etc., see [54]). But discussing the gain with such more complex transport models is beyond the scope of this paper and will certainly be tackled in further publications.

Let us now apply the above idea to our *fil rouge* problem in the next example 4.2.1: we revisit example 3.6.4 with a quickly built transport based learning algorithm.

Example 4.2.1 (Two local minimum separated by a large flat spot with transport) *In this last example, we consider the same situation as in example (3.6.4) for which we had a large flat spot between the two local minima. This time, instead of solving a diffusion equation by choosing particular $\alpha, \theta^* - \theta$ as in the previous example, we suggest choosing particular F, σ which will, in regime (57), solve a particular diffusion equation.*
We suggest considering equation

$$\partial_t u(t,\mathbf{v},\theta) + F(t,\mathbf{v},\theta)\cdot\partial_\theta u(t,\mathbf{v},\theta) + \sigma(t,\theta)u(t,\mathbf{v},\theta) = \sigma(t,\theta)\int P(t,\theta,\mathbf{v},\mathbf{v}')u(t,\theta,\mathbf{v}')\,\mathrm{d}\mathbf{v}', \tag{62}$$

in which

- $F(t,\mathbf{v},\theta) = F(t,\theta)\mathbf{v}$, *with* $\mathbf{v} = v\omega$. *The term* $F(t,\theta)v$ *is a velocity.*

[33] The set of choices is not unique, other possibilities can be considered, the following ones are only stated as examples.

– *Besides, ω is a direction in $\mathbb{S}^n$ (i.e. in the unit sphere such that $\omega \in \mathbb{R}^n$ and $|\omega| = 1$). Of course, in this example, $n = 1$ and $\mathrm{d}\omega = \frac{1}{2}\mathbf{1}_{[-1,1]}(\omega)\,\mathrm{d}\omega$ so that $\int \omega\,\mathrm{d}\omega = 0$ and $\int \omega \otimes \omega\,\mathrm{d}\omega = \int \omega^2\,\mathrm{d}\omega = \frac{1}{3}$, but ω can easily be generalised to arbitrary dimensions.*
This particular form for F makes sure we have

$$\begin{aligned}\forall (t,\theta) \in [0,T] \times \Theta, \int F(t,\theta,\mathbf{v})\,\mathrm{d}\mathbf{v} &= \int F(t,\theta) v\omega\,\mathrm{d}v\,\mathrm{d}\omega, \\ &= \iint F(t,\theta) v\omega\,\mathrm{d}v\,\mathrm{d}\omega, \\ &= F(t,\theta)\int v\,\mathrm{d}v \underbrace{\int \omega\,\mathrm{d}\omega}_{=0} = 0,\end{aligned}$$

as necessary (see appendix A),

– *The terms of the collisional part of (62), $\sigma(t,\theta), P(t,\theta,\mathbf{v},\mathbf{v}')$, are the total and scattering cross-sections/opacities.*

We want to choose the above quantities such that asymptotically in a specified regime defined by $\delta \sim 0$, (62) degenerates toward a diffusion equation given by

$$\partial_t \overline{u}(t,\theta) + \partial_\theta \left[\frac{F(t,\theta)v}{3\sigma(t,\theta)} \partial_\theta \overline{u}(t,\theta)\right] = 0. \tag{63}$$

Let us now try to build a diffusion coefficient $\frac{F(t,\theta)v}{3\sigma(t,\theta)}$ adapted to a learning problem:

– *First, let us set $\alpha = -\partial_\theta u(t,X,\theta)$ just as GD, SGD, dropout, in order to get:*

$$\begin{aligned}\int \alpha L'(t,X,\theta)(\theta^* - \theta)\,\mathrm{d}\mathcal{P}_X &= -\int \partial_\theta u(t,X,\theta) L'(t,X,\theta)(\theta^* - \theta)\,\mathrm{d}\mathcal{P}_X, \\ &= -\int J'(t,\theta)(\theta^* - \theta)\,\mathrm{d}\mathcal{P}_X.\end{aligned}$$

– *Now, by identification, this leads to*

$$\begin{aligned}\frac{F(t,\theta)v}{3\sigma(t,\theta)} &= -J'(t,\theta)\int (\theta^* - \theta)\,\mathrm{d}\mathcal{P}_X, \\ -\frac{F(t,\theta)v}{3J'(t,\theta)\int (\theta^* - \theta)\,\mathrm{d}\mathcal{P}_X} &= \sigma(t,\theta).\end{aligned}$$

From the above expression, either we choose F, σ and $\theta^* - \theta$ is set or we choose $\theta^* - \theta$ and F in order to set σ.

– *Let us set*

$$\theta^* - \theta = -\frac{vF(t,\theta)}{3J'(t,\theta)\left[\frac{1}{|J'(t,\theta)|} + \sigma_{acc}\right]}.$$

This choice (but many others could be done) makes sure having a positive cross-section σ having the general form

$$\begin{aligned}\sigma(t,\theta) &= \sigma_{grad}(t,\theta) + \sigma_{acc}, \\ &= \frac{1}{|J'(t,\theta)|} + \sigma_{acc} \text{ where } \sigma_{acc} \text{ is a parameter.}\end{aligned}$$

– *Furthermore, we choose*

$$F(t,\theta) \quad = \frac{1}{\sqrt{|J'(t,\theta)|}}.$$

This choice may appear unconventional at this stage as it means that the closer to zero the gradient, the higher the velocity. But this astonishing choice will be justified few lines below.

– *Let us now tackle the scattering term P. We choose it such that:*

$$\begin{aligned} P(t,\theta,\mathbf{v},\mathbf{v}')\,\mathrm{d}\mathbf{v}' &= P(t,\theta,v\omega,v'\omega')\,\mathrm{d}v'\,\mathrm{d}\omega', \\ &= \begin{bmatrix} +\dfrac{\sigma_{grad}(t,\theta)}{\sigma_{grad}(t,\theta)+\sigma_{acc}} & \delta_{v_{\min}}(v') & \delta_0\left(-\dfrac{J'(t,\theta)}{|J'(t,\theta)|}-\omega'\right) \\ +\dfrac{\sigma_{acc}}{\sigma_{grad}(t,\theta)+\sigma_{acc}} & \delta_{v_{\max}}(v') & \frac{1}{2}\mathbf{1}_{[-1,1]}(\omega') \end{bmatrix} \mathrm{d}v'\,\mathrm{d}\omega'. \end{aligned}$$

In P, the scattering term, $v \in \{v_{\min}, v_{\max}\}$ where $v_{\min}, v_{\max}$ are parameters. Of course, a continuous velocity distribution (just as for physical phenomena) could be built but this will be explored in further publications.

The above choices were certainly quite abruptly introduced. Let us justify them in the next lines:

– *σ has been chosen such that $\sigma(t,\theta) = \mathcal{O}(\frac{1}{\delta^2})$ when $|J'(\theta)| = \mathcal{O}(\delta^2) \to 0$. Having $\sigma \sim \frac{1}{\delta^2} \sim \frac{1}{|J'(\theta)|}$ ensures we will have a small mean time between two collisions in the vicinities where $J'(\theta) \sim \delta^2 \sim 0$.*

– *F has been chosen such that $F(t,\mathbf{v},\theta) = \frac{\mathbf{v}}{\sqrt{|J'(\theta)|}} = \mathcal{O}(\frac{1}{\delta})$ when $|J'(\theta)| = \mathcal{O}(\delta^2) \to 0$. Having $F \sim \frac{1}{\delta}$ (together with $\int F = 0$, see appendix A) makes sure a diffusion limit exists for the transport equation of interest. It is then given by (63).*

$\Longrightarrow$ *The two above choices, such that $F \sim \frac{1}{\delta}$ and $\sigma \sim \frac{1}{\delta}$ as $|J'| \sim \delta^2 \sim 0$, ensures the diffusion limit (63) of (62) will be valid in the vicinities of vanishing gradients.*

– *P is quite an arbitrary parameter, at least with respect to the diffusion limit and the regime characterised by $\delta \sim 0$. It has been chosen so that it behaves as a reaction kernel with two reactions, reaction 'grad' and reaction 'acc', and is inspired from what can be found in neutronics for example (see [25] for example):*

  – *reaction 'grad' occurs with probability $\frac{\sigma_{grad}(t,\theta)}{\sigma_{grad}(t,\theta)+\sigma_{acc}}$. It is less and less probable as $|J'| \sim 0$. Still, when reaction grad occurs, the velocity of the stochastic process θ_t is $v_{\min}$ and its direction ω' is the opposite to the direction of the gradient.*
  – *Reaction 'acc' occurs with probability $\frac{\sigma_{acc}}{\sigma_{grad}(t,\theta)+\sigma_{acc}}$. It is more and more probable as $|J'|$ comes close to zero: the stochastic process has reached a vicinity in which the gradient is under a certain **acc**uracy and a jump is triggered with an immediate increase of velocity to $v_{\max}$ together with an isotropic change of direction. The stochastic process is kind of teleported further away from the sufficiently explored vicinity.*

Solving (62) can be done applying the material the material of [60] (see for example how to built the non-analog MC scheme). We do not detail the construction of the stochastic process solving (62), and capturing the limit (63), but we insist on the fact that the resolution does not need the computation of J''.

Now, if we come back to our transport equation, we can certainly imagine even more relevant choices of F, σ, P which could help solve our problem. But let us focus on the choices made above:

figure 9 presents the results obtained with the non-analog MC scheme (see [60]) solving (62) for functional (53) with the flat spot between the two local minima (top pictures) and on a DNN architecture (bottom pictures).

Let now comment on the behaviour of the algorithm observable on figure 9:

- *once initialised, the algorithm takes stochastic time steps τ sampled from an exponential law, i.e. $\tau = -\frac{\log(\mathcal{U})}{\sigma(\theta_n)}$ with $\mathcal{U} \sim \mathcal{U}([0,1])$. Those are $\tau \sim \delta^2$ small if $J' \sim \delta \sim 0$ and larger otherwise.*
- *If $J' \sim \varepsilon \ll 1$, the probability of triggering an 'acc' reaction becomes more important: the process has an important probability of jumping to another location.*
- *Reaction kernel P ensures not spending too much time in the vicinity of a local minimum, once the gradient under a certain chosen threshold ε, and jumping more or less far from it.*
- *We insist the MC resolution of the linear transport equation we solve here is inconditionally stable, see [60].*
- *Finally, we insist this is an* exploration *algorithm, more than an optimisation one. This means that we must keep in memory the vector θ_{n^*} for iteration n^* such that $\overline{J}(\theta_{n^*}) < \overline{J}(\theta_n)$ because nothing ensures the error will keep going down as $n > n^*$.*

The jumps of the simulated stochastic process on figure 9 are observable for both applications on the pictures of the left column. The steps are further apart in the vicinities of big gradients and closer where the gradient comes close to zero. On the right column of figure 9, we can see that the process never spends too much times (iterations) in the vicinities of the minima: a jump is triggered once a small gradient vicinity explored. Let us now consider some more quantitative results in example 4.2.2.

Example 4.2.2 (A large flat spot with transport, quantitative results) *The results of example 4.2.1 related to figure 9 were mainly qualitative. We here suggest some quantitative ones. The conditions are the same as in the previous (quantitative) examples 3.1.2–3.1.4–3.5.2–3.6.2–4.2.2.*

The probabilities for the (penalised or not) transport algorithm to reach the vicinities of local/-global minima $\theta^* = -10$ and $\theta^* = 6$ are displayed in table 9. In both cases (penalised or not), we have a very important probability (98.75% and 97.58%) of recovering the vicinity of the global minimum $\theta^* = 6$. The probability of recovering the vicinity of the local minimum $\theta^* = -10$ is negligible and the probability of being outside the interval of interest remains small (1.25% and 2.42%). The probability for the solution to live in an even narrower vicinity $[6 - 10^{-4}, 6 + 10^{-4}]$ around the global minima is still very important ($\approx 98.76\%$). With this transport based algorithm, we can even try to reach narrower regions such as $[6 - 10^{-6}, 6 + 10^{-6}]$ around the global minima: the last line of table 9 shows that the algorithm still presents a 27.07% probability of scoring within this narrow region.

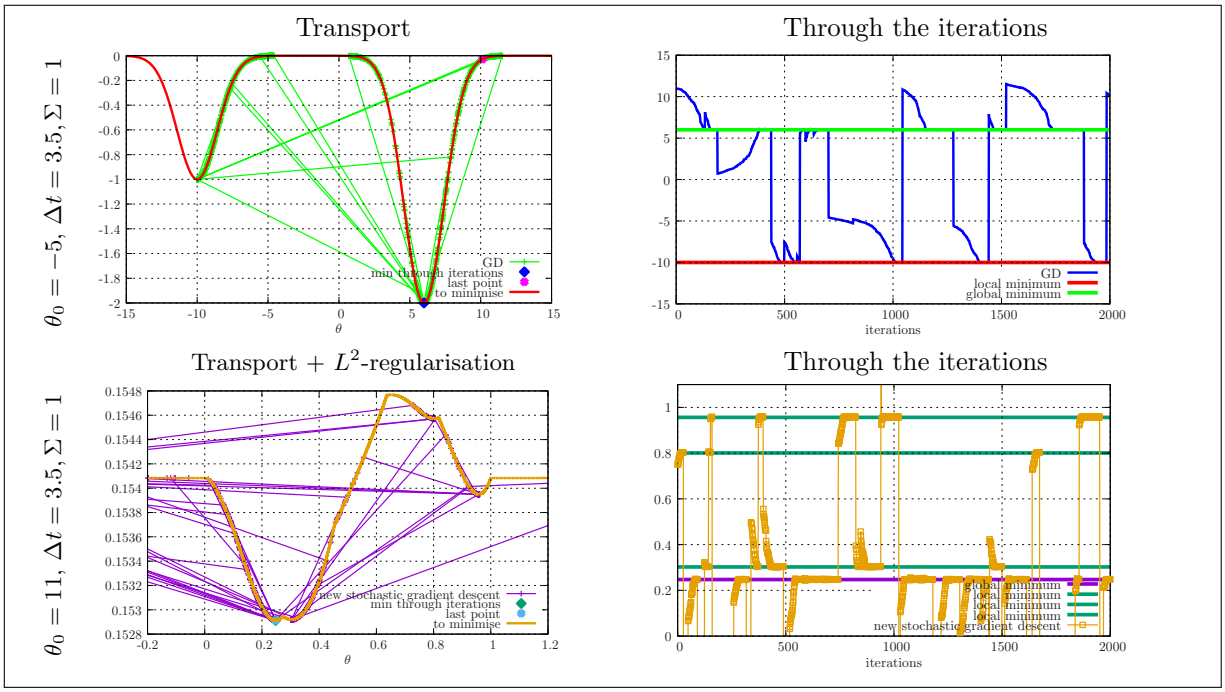

Figure 9: Top line: the figure revisit the test-case of example 3.6.4 but with the transport process solving (62). Bottom line: a cut along a certain direction of loss function used in order to tune a DNN. The DNN has 3 layers of one ReLU neuron except for the last layer which is linear, the loss function is the L^2 one with L^2 penalisation with $\xi = 10^{-7}$, and the architecture aims at approximating the Runge function. We choose $\sigma_{acc} = \frac{1}{\varepsilon}$ with (top) $v_{\min} = 10^3, v_{\max} = 10^8$, $\varepsilon = 10^{-5}$, (bottom= $v_{\min} = 10^3, v_{\max} = 10^8$, $\varepsilon = 10^{-7}$.

Initialiser $\theta_0 \sim \mathcal{U}([-15, 15])$	Vicinity of $\theta^* = 0$	Vicinity of $\theta^* = 6$
Transport SGD ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10 - 10^{-2}, -10 + 10^{-2}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-2}, 6 + 10^{-2}])$
As in example 4.2.1	$0.0\% \in [0.0\%, 0.0\%]$	$98.75\% \in [98.53\%, 98.97\%]$
$\xi = 10^{-5}$ + as in example 4.2.1	$0.0\% \in [0.0\%, 0.0\%]$	$97.58\% \in [97.28\%, 97.88\%]$
Transport SGD ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10 - 10^{-4}, -10 + 10^{-4}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-4}, 6 + 10^{-4}])$
As in example 4.2.1	$0.0\% \in [0.0\%, 0.0\%]$	$98.76\% \in [98.54\%, 98.98\%]$
Transport SGD ($N_{seed} = 10^4$)	$\mathbb{P}(\theta_{\min} \in [-10 - 10^{-6}, -10 + 10^{-6}])$	$\mathbb{P}(\theta_{\min} \in [6 - 10^{-6}, 6 + 10^{-6}])$
As in example 4.2.1	$0.0\% \in [0.0\%, 0.0\%]$	$27.07\% \in [26.20\%, 27.94\%]$

Table 9: Probabilities for the (penalised or not) transport based SGD algorithm to recover the local ($\theta^* = -10$) or the global ($\theta^* = 6$) minimum of (53). To compute the probabilities, we have resort to $N_{seed} = 10^4$ multiple initialisations with $\theta_0 \sim \mathcal{U}([-15, 15])$. Confidence intervals for the results are provided. The 0.00% are not significative but attest for a small probability. Comments are provided in example 4.2.2.

5 Conclusion

Let us finish by concluding remarks. The present document aims at helping numericians and people familiar with partial differential equations (PDEs) understanding how the most classical machine learning (ML) algorithms are built. The basic desired properties of optimisers are illustrated. An original (PDE based) framework built in order to design some new ML algorithms is suggested. Several classical ML algorithms are rewritten, reinterpreted in this PDE framework, which attests for its

generality. Of course, many ML algorithms from the dense and furnished literature are not tackled. From the PDE based framework we defined in this document, ML optimisation codes can be compared to instrumented Monte-Carlo codes solving drift-diffusion equations classically used in industrial contexts. The PDE based framework allows understanding, characterising (see the $\delta \sim 0$ discussions) and explaining when and why classical ML algorithms give satisfactory results despite some lacks of consistency. An analogy with transport and diffusion (based on the remark of [54]) is made and a new transport based algorithm is constructed and applied on simple examples. Some efforts remain to be done for those algorithms to be generalised and developped in a more classical ML context. Still, the results are statistically significative and promising enough for counting the design of new transport based ML algorithms amongst the perspectives of this work.

Acknowledgments

This document has been mainly written during the first lockdown of the covid-19 pandemic. The authors would like to thank Gilles Kluth for valuable discussions about similarities between MC codes and ML frameworks and Nestor Demeure for its careful reading and verification of the numerical examples.

References

[1] A. Bernede and G. Poëtte. An Unsplit Monte-Carlo solver for the resolution of the linear Boltzmann equation coupled to (stiff) Bateman equations. *Journal of Computational Physics*, 354:211 – 241, 2018.

[2] Andrew R. Barron. Universal Approximation Bounds for Superpositions of a Sigmoidal Function. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 39(3):930–945, may 1993.

[3] American Society of Mechanical Engineers ASME V&V 20-2009. Standard for Verification and Validation in Computational Fluid Dynamics and Heat Transfer. *ASME*, 2009.

[4] Peter Auer, Mark Herbster, and Manfred K. Warmuth. Exponentially many local minima for single neurons. *Departement of computer science*, ?

[5] François Bachoc. *Estimation paramétrique de la fonction de covariance dans le modèle de Krigeage par processus Gaussiens : application à la quantification des incertitudes en simulation numérique*. PhD thesis, 2013. Thèse de doctorat dirigée par Garnier, Josselin Mathématiques appliquées Paris 7 2013.

[6] C. Benedetti, A. Sgattoni, G. Turchetti, and P. Londrillo. `ALaDyn`: A High-Accuracy PIC Code for the Maxwell Vlasov Equations. *IEEE Transactions on Plasma Science*, 36(4):1790–1798, Aug 2008.

[7] J.P. Boyd. *Chebyshev and Fourier Spectral Methods: Second Revised Edition*. Dover Books on Mathematics. Dover Publications, 2001.

[8] H. Brezis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Universitext. Springer New York, 2010.

[9] E. Brun, F. Damian, C.M. Diop, E. Dumonteil, F.X. Hugot, C. Jouanne, Y.K. Lee, F. Malvagi, A. Mazzolo, O. Petit, J.C. Trama, T. Visonneau, and A. Zoia. Tripoli-4, cea, edf and areva reference monte carlo code. *Annals of Nuclear Energy*, 82:151 – 160, 2015. Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013, SNA + MC 2013. Pluri- and Trans-disciplinarity, Towards New Modeling and Numerical Simulation Paradigms.

[10] Emeric Brun, Stéphane Chauveau, and Fausto Malvagi. Patmos: A prototype monte carlo transport code to test high performance architectures. In Proceedings of International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering, Jeju, Korea, 2017.

[11] Kevin Burrage and Tianhai Tian. A note on the stability properties of the euler methods for solving stochastic differential equations. *New Zealand J. Math.*, 29, 10 1999.

[12] Jose Antonio Carrillo and Mattia Zanella. Monte Carlo gPC methods for diffusive kinetic flocking models with uncertainties. 2019. preprint.

[13] J. Castor. *Radiation hydrodynamics*. Cambridge University Press, 2004.

[14] F. Chaland and G. Samba. Discrete ordinates method for the transport equation preserving one-dimensional spherical symmetry in two-dimensional cylindrical geometry. *Nuclear Science and Engineering*, 182(4):417–434, 2016.

[15] S. Chapman and T.G. Cowling. *The mathematical theory of non-uniform gases: An account of the kinetic theory of viscosity, thermal conduction, and diffusion in gases*. Cambridge University Press, 1960.

[16] Mathew A. Cleveland and Nick Gentile. Mitigating teleportation error in frequency-dependent hybrid implicit monte carlo diffusion methods. *Journal of Computational and Theoretical Transport*, 43(1-7):6–37, 2014.

[17] J.-F. Clouet and G. Samba. Asymptotic diffusion limit of the symbolic monte-carlo method for the transport equation. *Journal of Computational Physics*, 195(1):293 – 319, 2004.

[18] Xiaowu Dai and Yuhua Zhu. Toward Theoretical Understanding of Large Batch Training in Stochastic Gradient Descent. *stat.ML*, 2018.

[19] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2933–2941. Curran Associates, Inc., 2014.

[20] J. D. Densmore and E. W. Larsen. Asymptotic equilibrium diffusion analysis of time-dependent Monte Carlo methods for grey radiative transfer. *Journal of Computational Physics*, 199:175–204, September 2004.

[21] Bruno Després, Gaël Poëtte, and Didier Lucor. *Robust Uncertainty Propagation in Systems of Conservation Laws with the Entropy Closure Method*, volume 92 of *Lecture Notes in Computational Science and Engineering*. Uncertainty Quantification in Computational Fluid Dynamics, 2013.

[22] Jakob Drrwchter, Thomas Kuhn, Fabian Meyer, Louisa Schlachter, and Florian Schneider. A hyperbolicity-preserving discontinuous stochastic galerkin scheme for uncertain hyperbolic systems of equations. *Journal of Computational and Applied Mathematics*, 370:112602, May 2020.

[23] V.V. Fedorov. *Theory Of Optimal Experiments*. Probability and Mathematical Statistics. Elsevier Science, 1972.

[24] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[25] S. Glasstone G.I. Bell. *Nuclear Reactor Theory*. Van Nostrand Reinhold Company, New York, N.Y. 10001, 1970.

[26] F. Golse and G. Allaire. *Transport et Diffusion*. 2015. Polycopié de cours.

[27] François Golse. *The Boltzmann equation and its hydrodynamic limits*, volume 2 of *Evolutionary equations*. 2005.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[29] D. Hilbert. Begrundung der kinetischen gastheorie. *Math. Ann.*, 72:562–577, 1912.

[30] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Netw.*, 4(2):251–257, March 1991.

[31] Stanislaw Jastrzkebski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos J. Storkey. Three factors influencing minima in SGD. *CoRR*, abs/1711.04623, 2017.

[32] Daniel Kahneman. *Thinking, fast and slow*. Farrar, Straus and Giroux, New York, 2011.

[33] Gilles Kluth. *Analyse mathmatique et numrique de systmes hyperlastiques et introduction de la plasticit*. PhD thesis, 2008. Thse de doctorat dirige par Desprs, Bruno et Frey, Pascal Mathmatiques appliques Paris 6 2008.

[34] Gilles Kluth and Bruno Després. 2d finite volume lagrangian scheme in hyperelasticity and finite plasticity. In Gunilla Kreiss, Per Lötstedt, Axel Målqvist, and Maya Neytcheva, editors, *Numerical Mathematics and Advanced Applications 2009*, pages 489–496, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[35] J. Kusch, G. W. Alldredge, and M. Frank. Maximum-principle-satisfying second-order intrusive poly- nomial moment scheme. *arXiv preprint arXiv:1712.06966*, 2017.

[36] J. Kusch and M. Frank. Intrusive methods in uncertainty quantification and their connection to kinetic theory. *International Journal of Advances in Engineering Sciences and Applied Mathematics*, pages 1–16, 2018.

[37] Jonas Kusch, Ryan G. McClarren, and Martin Frank. Filtered stochastic galerkin methods for hyperbolic equations. *J. Comput. Phys.*, 2018.

[38] Jonas Kusch, Jannick Wolters, and Martin Frank. Intrusive acceleration strategies for uncertainty quantification for hyperbolic systems of conservation laws. *Journal of Computational Physics*, 419:109698, 2020.

[39] B. Lapeyre, E. Pardoux, and R. Sentis. *Méthodes de Monte Carlo pour les équations de transport et de diffusion*. Number 29 in Mathématiques & Applications. Springer-Verlag, 1998.

[40] E. E. Lewis and W. F. Miller Jr. *Computational Methods of Neutron Transport*. John Wiley and Son New York, 1984.

[41] Yeong-Chenga Liou, Chunmei Shi, Yu Xiao, and Chiping Zhang. The convergence and ms stability of exponential euler method for semilinear stochastic differential equations. *Abstract and Applied Analysis*, 2012.

[42] R. B. Lowrie, J. E. Morel, and J. A. Hittinger. The coupling of radiation and hydrodynamics. *Astrophysical Journal*, 521:432–450, August 1999.

[43] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. *CoRR*, abs/1709.02540, 2017.

[44] P. Maire, R. Abgrall, J. Breil, and J. Ovadia. A cell-centered lagrangian scheme for two-dimensional compressible flow problems. *SIAM Journal on Scientific Computing*, 29(4):1781–1824, 2007.

[45] P.H. Maire. *Contribution to the numerical modeling of Inertial Confinement Fusion*. Habilitation à diriger des recherches, Université Bordeaux I, February 2011.

[46] A. Majda. *Compressible Fluid Flow and Systems of Conservation Laws*. Applied Mathematical sciences, 53, New York, springer-verlag edition, 1984.

[47] B.G. Malkiel. *A Random Walk Down Wall Street: The Time-Tested Strategy for Successful Investing (Eleventh Edition)*. W. W. Norton, 2015.

[48] Michael Scott McKinley, Eugene D. Brooks III, and Abraham Szoke. Comparison of implicit and symbolic implicit monte carlo line transport with frequency weight vector extension. *Journal of Computational Physics*, 189(1):330 – 349, 2003.

[49] Hossein Mobahi. Training recurrent neural networks by diffusion. *CoRR*, abs/1601.04114, 2016.

[50] Paul Novello, Gaël Poëtte, David Lugato, and Pietro M Congedo. Explainable Hyperparameters Optimization using Hilbert-Schmidt Independence Criterion. working paper or preprint, February 2021.

[51] Paul Novello, Gaël Poëtte, David Lugato, and Pietro Marco Congedo. A Taylor Based Sampling Scheme for Machine Learning in Computational Physics, December 2019. Second Workshop on Machine Learning and the Physical Sciences (NeurIPS 2019), Vancouver, Canada.

[52] Paul Novello, Gaël Poëtte, David Lugato, and Pietro Marco Congedo. Variance Based Samples Weighting for Supervised Deep Learning. working paper or preprint, January 2021.

[53] Bernt Oksendal. *Stochastic Differential Equations (3rd Ed.): An Introduction with Applications*. Springer-Verlag, Berlin, Heidelberg, 1992.

[54] G. C. Papanicolaou. Asymptotic Analysis of Transport Processes. *Bulletin of the American Mathematical Society*, 81(2), 1975.

[55] Lorenzo Pareschi. An introduction to uncertainty quantification for kinetic equations and related problems, 2020.

[56] Grigorios A Pavliotis. *Stochastic Processes and Applications Diffusion Processes, the Fokker-Planck and Langevin Equations*. Springer, 2014.

[57] G. Poëtte. Spectral convergence of the generalized Polynomial Chaos reduced model obtained from the uncertain linear Boltzmann equation. *Preprint submitted to Mathematics and Computers in Simulation*, 2019.

[58] Gaël Poëtte. A comparative study of generalized Polynomial Chaos based Approximations: integration vs. regression vs. collocation vs. kriging. working paper or preprint, July 2018.

[59] Gaël Poëtte. A gPC-intrusive Monte-Carlo scheme for the resolution of the uncertain linear Boltzmann equation. *Journal of Computational Physics*, 385:135 – 162, 2019.

[60] Gaël Poëtte. *Contribution to the mathematical andnumerical analysis of uncertain systemsof conservation laws and of the linear and nonlinear boltzmann equation*. Habilitation à diriger des recherches, University of Bordeaux, September 2019.

[61] Gaël Poëtte. Efficient uncertainty propagation for photonics: combining Implicit Semi-analog Monte Carlo (ISMC) and Monte Carlo generalised Polynomial Chaos (MC-gPC). working paper or preprint, November 2020.

[62] Gaël Poëtte and Emeric Brun. Efficient uncertain k eff computations with the Monte Carlo resolution of generalised Polynomial Chaos Based reduced models. working paper or preprint, November 2020.

[63] Gal Potte and Xavier Valentin. A new implicit monte-carlo scheme for photonics (without teleportation error and without tilts). *Journal of Computational Physics*, 412:109405, 2020.

[64] L. Saint Raymond. *The Boltzmann equation and its hydrodynamic limits*, volume 1971 of *Lecture Notes in Mathematics*. Springer-Verlag Berlin Heidelberg, 2009.

[65] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[66] L. Schlachter and F. Schneider. A hyperbolicity-preserving stochastic galerkin approximation for uncertain hyperbolic systems of equations. *arXiv preprint arXiv:1710.03587*, 2017.

[67] Louisa Schlachter, Florian Schneider, and Oliver Kolb. Weighted essentially non-oscillatory stochastic galerkin approximation for hyperbolic conservation laws. *Journal of Computational Physics*, 419:109663, 2020.

[68] Richard P. Smedley-Stevenson and Ryan G. McClarren. Asymptotic diffusion limit of cell temperature discretisation schemes for thermal radiation transport. *Journal of Computational Physics*, 286:214 – 235, 2015.

[69] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[70] D.W. Stroock and S.R.S. Varadhan. *Multidimensional Diffusion Processes*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 1997.

[71] Emmanuel Vazquez. *Modélisation comportementale de systèmes non-linéaires multivariables par méthodes à noyaux et applications*. PhD thesis, Paris 11, 2005. Thèse de doctorat dirigée par Walter, Éric Sciences appliquées.

[72] Allan B. Wollaber. Four decades of implicit monte carlo. *Journal of Computational and Theoretical Transport*, 45(1-2):1–70, 2016.

[73] Mattia Zanella. Structure preserving stochastic galerkin methods for fokker-planck equations with background interactions, 2019.

A Formal proof of transport equation degenerating toward the diffusion equation

Plugging the Hilbert development $u = u_0 + u_1\delta + u_2\delta^2 + \mathcal{O}(\delta^3)$ defined in section 4.1 into (55) yields

$$\delta^2\partial_t(u_0 + \delta u_1 + \delta^2 u_2) + \delta F\partial_x(u_0 + \delta u_1 + \delta^2 u_2) + \sigma(x,t)(u_0 + \delta u_1 + \delta^2 u_2) = \\ \sigma(x,t)\int P(x,t,v,v')(u_0 + \delta u_1 + \delta^2 u_2)\,\mathrm{d}v' + \mathcal{O}(\delta^5).$$

Let us identify the terms in factor of $1, \delta, \delta^2$ and cancel them. This ensures having

$$\begin{cases} u_0(x,t,v) = \displaystyle\int P(x,t,v,v')u_0(x,t,v')\,\mathrm{d}v', \\ v\partial_x u_0(x,t,v) = -\sigma(x,t)u_1(x,t,v) + \sigma(x,t)\displaystyle\int P(x,t,v,v')u_1(x,t,v')\,\mathrm{d}v', \\ \partial_t u_0(x,t,v) + v\partial_x u_1(x,t,v) + \sigma(x,t)u_2(x,t,v) = \sigma(x,t)\displaystyle\int P(x,t,v,v')u_2(x,t,v')\,\mathrm{d}v'. \end{cases} \tag{64}$$

The first line of the above equation $(I-K)u_0 = 0 = u_0(x,t,v) - \int P(x,t,v,v')u_0(x,t,v')\,\mathrm{d}v'$ tells that, cf. theorem 3 (Fredholm's alternative), $u_0(x,t,v) = u_0(x,t)$ is independent of v (constant function are in the kernel of operator $I-K$).

Theorem 3 (Fredholm's alternative [26]) *Consider operator*

$$(I-K)f(x,t,v) = f(x,t,v) - \int \pi(x,t,v,v')f(x,t,v')\,\mathrm{d}v',$$

where π satisfies $\int \pi(x,t,v,v')\,\mathrm{d}v' = 1, \forall x,t,v$ and $\int \pi(x,t,v,v')\,\mathrm{d}v = 1, \forall x,t,v'$. The kernel of $I-K$ is given by $ker(I-K) = \{f(x,t,v) = f(x,t), i.e.$ constant function with respect to $v\}$.

Furthermore, a solution f of

$$(I-K)f(x,t,v) = S(x,t,v), \tag{65}$$

exists if and only if $\int S(x,t,v)\,\mathrm{d}v = 0$. Besides, if f is solution, then $f - \int f\,\mathrm{d}v$ is also solution so that the uniqueness comes from the fact that it exists a unique solution f such that $\int f\,\mathrm{d}v = 0$. The unique solution of (65) has the form $f(x,v,v) = f_0(x,t,v) + C_1(x,t)$.

With the result given by Fredholm's alternative (theorem 3), i.e. the fact that $u_0(x,t,v) = u_0(x,t)$, (64) becomes

$$\begin{cases} u_0(x,t,v) = u_0(x,t), \\ F(x,t,v)\partial_x u_0(x,t) = -\sigma(x,t)u_1(x,t,v) + \sigma(x,t)\displaystyle\int P(x,t,v,v')u_1(x,t,v')\,\mathrm{d}v', \\ \partial_t u_0(x,t) + F(x,t,v)\partial_x u_1(x,t,v) + \sigma(x,t)u_2(x,t,v) = \sigma(x,t)\displaystyle\int P(x,t,v,v')u_2(x,t,v')\,\mathrm{d}v'. \end{cases}$$

The second line of the above equation can be recast as

$$-\frac{F(x,t,v)}{\sigma(x,t)}\partial_x u_0(x,t) = (I-K)u_1(x,t,v). \tag{66}$$

Once again, theorem 3 ensures that the solution exists as by hypothesis, $\int F(x,t,v)\,\mathrm{d}v = 0, \forall x,t$ (otherwise, no solution). Besides, according to theorem 3, the solution u_1 is unique and has the form[34]

$$u_1(x,t,v) = C_1(x,t) - b(x,t,v)\partial_x u_0(x,t),$$

where C_1 satisfies $(I-K)C_1 = 0$, and $b(x,t,v)$ is such that

$$\begin{cases} (I-K)b(x,t,v) = +\frac{F(x,t,v)}{\sigma(x,t)}, \\ \displaystyle\int b(x,t,v')\,\mathrm{d}v' = 0. \end{cases} \tag{67}$$

We then have

$$\begin{cases} u_0(x,t,v) = u(x,t), \\ u_1(x,t,v) = C_1(x,t) - b(x,t,v)\partial_x u_0(x,t), \\ \partial_t u_0(x,t) + F(x,t,v)\partial_x u_1(x,t,v) + \sigma(x,t)u_2(x,t,v) = \sigma(x,t)\displaystyle\int P(x,t,v,v')u_2(x,t,v')\,\mathrm{d}v'. \end{cases}$$

The last equation has once again the form $(I-K)u_2 = g$ with

$$g(x,t,v) = -\tfrac{1}{\sigma(x,t)}\partial_t u_0(x,t) - \tfrac{F(x,t,v)}{\sigma(x,t)}\partial_x u_1(x,t,v) = u_2(x,t,v) - \int P(x,t,v,v')u_2(x,t,v')\,\mathrm{d}v'.$$

[34] It is easy verifying it is a solution of (66). The uniqueness comes from both theorem 3 and the constraint in (67).

Let us apply Fredholm alternative. For u_2 to exist, we must have

$$\int g(x,t,v)\,\mathrm{d}v = 0 = -\frac{1}{\sigma(x,t)}\partial_t u_0(x,t) - \frac{1}{\sigma(x,t)}\int F(x,t,v)\partial_x u_1(x,t,v)\,\mathrm{d}v.$$

Furthermore, we have

$$\begin{aligned}\int F(x,t,v)\partial_x u_1(x,t,v)\,\mathrm{d}v &= \int F(x,t,v)\partial_x\left[C_1(x,t) - b(x,t,v)\partial_x u_0(x,t)\right]\,\mathrm{d}v,\\ &= \underbrace{\left[\int F(x,t,v)\,\mathrm{d}v\right]}_{=0}\partial_x C_1(x,t) - \partial_x\left[\left(\int F(x,t,v)b(x,t,v)\,\mathrm{d}v\right)\partial_x u_0(x,t)\right],\\ &= -\partial_x\left[\left[\int F(x,t,v)b(x,t,v)\,\mathrm{d}v\right]\partial_x u_0(x,t)\right].\end{aligned}$$

Hence, u_2 exists if and only if

$$\int g(x,t,v)\,\mathrm{d}v = 0 = -\frac{1}{\sigma(x,t)}\partial_t u_0(x,t) + \frac{1}{\sigma(x,t)}\partial_x\left[\left[\int F(x,t,v)b(x,t,v)\,\mathrm{d}v\right]\partial_x u_0(x,t)\right].$$

The above line (already) implies

$$\partial_t u_0(x,t) - \partial_x\left[\left[\int F(x,t,v)b(x,t,v)\,\mathrm{d}v\right]\partial_x u_0(x,t)\right] = 0.$$

But let us check that u_2 remains in agreement with the above expression. To obtain u_2, we need to solve

$$\begin{aligned}&-\tfrac{1}{\sigma(x,t)}\partial_x\left[\left[\int F(x,t,v)b(x,t,v)\,\mathrm{d}v\right]\partial_x u_0(x,t)\right] - \tfrac{F(x,t,v)}{\sigma(x,t)}\partial_x C_1(x,t) + \tfrac{1}{\sigma(x,t)}\partial_x\left[F(x,t,v)b(x,t,v)\partial_x u_0(x,t)\right] =\\ &\qquad\qquad u_2(x,t,v) - \int P(x,t,v,v')u_2(x,t,v')\,\mathrm{d}v'.\\ &-\tfrac{F(x,t,v)}{\sigma(x,t)}\partial_x C_1(x,t) + \tfrac{1}{\sigma(x,t)}\partial_x\left[\left(F(x,t,v)b(x,t,v) - \int F(x,t,v)b(x,t,v)\,\mathrm{d}v\right)\partial_x u_0(x,t)\right] = (I-K)u_2(x,t,v).\end{aligned}$$

Let us now once again apply Fredholm's alternative. Let us introduce Σ such that

$$\begin{cases}(I-K)\Sigma(x,t,v) = F(x,t,v)b(x,t,v) - \displaystyle\int F(x,t,v)b(x,t,v)\,\mathrm{d}v,\\ \displaystyle\int \Sigma(x,t,v)\,\mathrm{d}v = 0,\end{cases}$$

and $C_2(x,t)$ such that $(I-K)C_2(x,t) = 0$. We obtain

$$u_2(x,t,v) = \frac{1}{\sigma(x,t)}\partial_x\left[\Sigma(x,t,v)\partial_x u_0(x,t)\right] - \frac{1}{\sigma(x,t)}b(x,t,v)\partial_x C_1(x,t) + C_2(x,t).$$

If we now replace u_1 and u_2 in the last equation of (64), we get the diffusion limit:

$$\begin{aligned}&\partial_t u_0(x,t) + F(x,t,v)\partial_x u_1(x,t,v) + \sigma(x,t)u_2(x,t,v) - \sigma(x,t)\int P(x,t,v,v')u_2(x,t,v')\,\mathrm{d}v' = 0,\\ &\partial_t u_0(x,t) + F(x,t,v)\partial_x\left[C_1(x,t) - b(x,t,v)\partial_x u_0(x,t)\right] + \sigma(x,t)u_2(x,t,v) = \int \sigma(x,t,v,v')u_2(x,t,v')\,\mathrm{d}v',\\ &\partial_t u_0(x,t) + F(x,t,v)\partial_x\left[C_1(x,t) - b(x,t,v)\partial_x u_0(x,t)\right] + \sigma(x,t)(I-K)u_2(x,t,v) = 0,\\ &\partial_t u_0(x,t) - F(x,t,v)\partial_x\left[b(x,t,v)\partial_x u_0(x,t)\right] + \partial_x\left[\left(F(x,t,v)b(x,t,v) - \int F(x,t,v)b(x,t,v)\,\mathrm{d}v\right)\partial_x u_0(x,t)\right] = 0,\\ &\partial_t u_0(x,t) - \partial_x\left[\left(\int F(x,t,v)b(x,t,v)\,\mathrm{d}v\right)\partial_x u_0(x,t)\right] = 0.\end{aligned}$$

This ends the proof.