A Decision-Making Machine Learning Approach in Hermite Spectral Approximations of Partial Differential Equations

Manzini, Gianmarco
Fatone, Lorella
Funaro, Daniele

To be published in: Journal of Scientific Computing

DOI to publisher's version: 10.1007/s10915-022-01853-4

A decision-making machine learning approach in Hermite spectral approximations of partial differential equations

L. Fatone, [c] D. Funaro [a,b] and G. Manzini [b]

[a] *Dipartimento di Scienze Chimiche e Geologiche, Università degli Studi di Modena e Reggio Emilia, Via campi 103, 41125 Modena Italy;*
[b] *Istituto di Matematica Applicata e Tecnologie Informatiche, Consiglio Nazionale delle Ricerche, via Ferrata 1, 27100 Pavia,*
[c] *Dipartimento di Matematica, Università degli Studi di Camerino, Via Madonna delle Carceri 9, 62032 Camerino, Italy;*

Abstract

The accuracy and effectiveness of Hermite spectral methods for the numerical discretization of partial differential equations on unbounded domains, are strongly affected by the amplitude of the Gaussian weight function employed to describe the approximation space. This is particularly true if the problem is under-resolved, i.e., there are no enough degrees of freedom. The issue becomes even more crucial when the equation under study is time-dependent, forcing in this way the choice of Hermite functions where the corresponding weight depends on time. In order to adapt dynamically the approximation space, it is here proposed an automatic decision-making process that relies on machine learning techniques, such as deep neural networks and support vector machines. The algorithm is numerically tested with success on a simple 1D problem, but the main goal is its exportability in the context of more serious applications.

Key words: time-dependent heat equation, generalized Hermite functions, machine learning, neural networks, support vector machine, spectral methods
1991 MSC: 65N35, 35Q83

1. Introduction

The aim of the present paper is to show how Machine Learning (ML) techniques can be employed, via a suitable implementation, in the field of spectral methods for PDEs, using Hermite polynomials as approximating functions. To ease the exposition, we approach this subject in an "academic way" by just considering the 1D time-dependent heat equation. However, we are motivated by precise needs emerging in the context of plasma physics, which we will explain in the paragraphs to follow. In the first part of this introduction, we provide a quick historical background to the role of Hermite functions as approximation basis for PDEs, and we postpone to the second part a brief discussion on the ML algorithms of our interest.

Generalized Hermite functions are constructed as $H_n w_\alpha$, where H_n is the n-th Hermite polynomial and $w_\alpha(x) = \exp(-\alpha^2(x-\beta)^2)$ is the weight function. We refer to α and β as the scaling and the shifting factors. A first convergence analysis of the Hermite spectral approximation of the 1D heat equation with both Galerkin and collocation methods was provided in [12], and for the Burgers equation in [16], by assuming $\alpha = 1$ and $\beta = 0$. Hermite approximations of functions and solutions to PDEs may however perform poorly when there is not a sufficient number of degrees of freedom, and inappropriate choices of the above parameters do not really help improving the outcome. For example, if $\alpha = 1$ and $\beta = 0$, the approximation of $\sin x$ in the interval $[-N, N]$ requires at least N^2 expansion terms [14, pp. 44–45]. Despite this very pessimistic anticipation, we known nowadays that more effective methods

can be obtained by choosing α and β carefully. As pointed out in [41], we only need a reasonable number of terms N in the truncated expansion if $\beta = 0$ and the scaling factor α is taken as $max_{1\le j\le N}\big(\xi_j^N\big)/N$, where ξ_j^N, $j = 1, \ldots, N$ are the roots of H_N. Such a significant improvement has reopened the way towards the design of Hermite spectral methods that are computationally efficient for solving physics and engineering problems. On the other hand, this poses the absolutely nontrivial problem of how to choose acceptable values for the scaling and shifting parameters.

More recently, generalized Hermite functions have been applied to solving time dependent problems, assuming that $\alpha = \alpha(t)$ and $\beta = \beta(t)$ may also change in time. Noteworthy developments in this direction are found for instance in [24, 23, 48]. Those papers cover a wide range of applications ranging from the discretization of linear and nonlinear diffusion and convection-diffusion equations, to the generalized Ginzburg–Landau equation and the Fokker-Planck equation.

It is important to mention that Hermite functions provide the most natural framework for the numerical approximations of the distribution function solving the Vlasov equation, which is the most basic mathematical model in noncollisional plasma physics [15]. Indeed, Hermite functions are directly linked to the Maxwellian distribution function that describes a noncollisional plasma close to the equilibrium state. When the plasma develops a strong non-Maxwellian behavior, an impractical number of Hermite basis functions could be needed, thus making a numerical solution too expensive. In [34], it was shown through numerical experiments that, even in bad situations, using weight functions of the form $\exp\big(-\alpha^2(x-\beta)^2\big)$ with a careful choice of α and β, it is possible to reduce significantly the computational burden by orders of magnitude. This fact was exploited in successive works for investigating the properties of Vlasov-based plasma models [1, 6] and developing computationally efficient numerical methods (see, e.g.: [10, 8, 9, 13, 26, 4], and the code implementations described in [25, 47]).

To the best of our knowledge, however, the selection of reasonable values for α and β (also depending on time) is still an open problem, since only partial, often unsatisfactory, answers have been given, sometimes limited to specific coefficients that must be somehow imposed by the user at the beginning of the simulation. Here, our contribution is finalized to the automatic detection of the most appropriate value of α, whereas, for simplicity β will remain equal to zero. The methodology is based on a ML approach that tries to recognize the shape of the numerical solution while it evolves in time, and select dynamically the proper value of α. The ML algorithm is initially trained by means of a suitable input set, where each one of its elements brings along the correct value of α. As the numerical solution advances in time, the so instructed ML algorithm evaluates periodically the most reasonable value of α at that time. The computation then continues with the updated value.

ML algorithms are nowadays widely applied in the numerical treatment of differential equations arising in science and engineering. Many of such learning algorithms have been designed in recent times for solving classification and regression problems, in the purpose of achieving better effectiveness and efficiency in tackling physics and engineering applications with computers [20]. A review of these developments is beyond the scope of this article, so we briefly recall some background material useful for the discussion of our problem, without any claim of being exhaustive. Neural Networks (NN) can be used to approximate sufficiently regular functions and their derivatives, as well as solutions to differential equations [3, 29, 27]. Moreover, NN can reduce the costs of modelling computational domains with complex geometries when solving forward and inverse problems [31], and can be used as a substitute to conventional constitutive material models [39]. A detailed introduction to these topics can be found in [49]. In [37], the solutions of a high-dimensional PDE are approximated through a deep NN, suitably trained to conform the behavior of a given differential operator (equipped with initial and boundary conditions). Physics-informed neural networks (PINN) are supervised learning algorithms that constraint a numerical solution to satisfy a given set of physics laws or PDEs, as studied for example in [30, 22, 19, 28, 36].

As an alternative to the techniques mentioned above, we may consider the Support Vector Machine (SVM) approach. Originally proposed at the beginning of the sixties [45, 43, 44], the SVM is a class of learning algorithms that can be used for solving classification and regression problems, after a training over a suitable set of input data [46]. The decision functions is a linear combination of a set of basis functions that are nonlinear and parameterized by the so called Support Vectors. This approach can also provide a computationally efficient function representation when the input space is high dimensional [17, 40, 42, 32]. A SVM is trained to minimize the approximation errors, by searching for a nonlinear function that fits the set of input data within an assigned threshold and is as flat as possible. The implementation is usually based on special polynomials or Gaussian kernels [18, 33]. An exhaustive review of SVM algorithms with their practical applications is beyond the scope of this paper, so we refer the interested reader to [38] and [7, 2] for a more detailed introduction to both theory and usage.

The scope of this paper is to implement a couple of ML methods of the type just described. As often happens in this kind of applications, one of the most crucial issues will be the detection of the appropriate training sets. The material is organized as follows. In Section 2, we briefly review the basic features of Hermite spectral method for the 1D heat equation on the straight-line. In Section 3 we discuss how the modification of α may impact on the scheme formulation. In Section 4 we reformulate the problem as a collocation method, and introduce a discretization in time admitting a scaling factor $\alpha = \alpha(t)$ changing with time. In Section 5 we design an automatic decision-making strategy for the dynamical determination of α, applied to the numerical resolution of the heat equation with homogeneous right-hand side. This is performed by training the ML learning algorithm with the help of Gaussian-like profiles. In Section 6 we apply a similar strategy to the non-homogeneous case by employing a training set containing suitable spline functions. In Section 7, we provide some final remarks and discuss future projects concerning the application of this ML strategy to the Vlasov equation.

2. Galerkin approximation of the heat equation

Let $\mathbb{R}$ be the set of real numbers. We work with the heat equation, that in strong form is given by:

$$\partial_t u - \partial_{xx} u = f, \qquad x \in \mathbb{R}, \quad t > 0, \tag{1a}$$

$$u(x,0) = u_0(x), \qquad x \in \mathbb{R}, \tag{1b}$$

where f is a given forcing term and u_0 is the initial guess. We assume that the solution $u = u(x,t)$ has an exponential decay at infinity, although we do not specify at the moment the exact decay rate. This crucial issue will be discussed as we proceed with our investigation. For this reason, most of the passages that follow are to be intended in informal way.

When $f = 0$, the exact solution of problem (1a)-(1b) is explicitly known:

$$u(x,t) = 2\alpha(t) w_\alpha(x,t) \quad \text{with } \alpha(t) = \frac{1}{2\sqrt{t+1}}, \ \ w_\alpha(x,t) = \exp\big(-(\alpha(t)x)^2\big). \tag{2}$$

As the reader can notice, the behavior of u for $x \to \pm\infty$ depends on time. This means that it is not easy to define an appropriate functional space as a natural habitat for u. In particular, we would like to approximate u in (2) by Gaussian functions that display a different decay rate (with α constant in time for instance). For most of the paper we will continue to play with the simplified case $f = 0$, in order to prepare the ground for the non-homogeneous case.

Let us go straight to the approximation. For a positive integer number N, let $\mathbb{P}_N$ denote the space of polynomials of degree at most N. Consider then the functional space:

$$V_N(t) := \Big\{ v_N(\cdot,t) = w_\alpha(\cdot,t)\phi_N \,|\, \phi_N \in \mathbb{P}_N \Big\}, \tag{3}$$

where the weight function is given in (2). The classical Galerkin approximation is obtained in the usual way through the following variational formulation:

For $t \in [0,T]$, find $u_N(t) \in V_N(t)$ such that:

$$\int_{\mathbb{R}} \frac{\partial u_N}{\partial t}\phi_N \, dx + \int_{\mathbb{R}} \frac{\partial u_N}{\partial x}\frac{\partial \phi_N}{\partial x} \, dx = \int_{\mathbb{R}} f\phi_N \, dx, \qquad \forall \phi_N \in \mathbb{P}_N, \tag{4}$$

$$\int_{\mathbb{R}} \big(u_N(x,0) - u_0(x)\big)\phi_N(x,0) \, dx = 0, \qquad \forall \phi_N \in \mathbb{P}_N. \tag{5}$$

The exponential decrease of the weight function $w_\alpha(x,t)$ for $x \to \pm\infty$ justifies the omission of the boundary terms in the integration by parts.

Before proceeding, we need to introduce the set of Hermite polynomials. In fact, we will expand the approximation u_N in the basis of Hermite functions. In the specific case, we first use Hermite polynomials multiplied by the weight function $w_\alpha(x,t)$ introduced in (2). Later on, we will examine other options.

We recall that Hermite polynomials are recursively defined by

$$H_0(\zeta) = 1, \qquad H_1(\zeta) = 2\zeta, \tag{6}$$

$$H_{n+1}(\zeta) = 2\zeta H_n(\zeta) - 2nH_{n-1}(\zeta), \qquad n \geq 2, \tag{7}$$

with the useful convention that $H_s(\zeta) = 0$ if $s < 0$ [35]. These polynomials are orthogonal with respect to the weighted inner product of $L^2(\mathbb{R})$ using the weight function w_α with α constantly equal to 1. In practice, we have:

$$\int_{\mathbb{R}} H_\ell(\zeta) H_m(\zeta) e^{-\zeta^2}\, d\zeta = 2^m\, m!\, \sqrt{\pi}\, \delta_{\ell,m}, \tag{8}$$

where $\delta_{\ell,m}$ is the Dirac delta symbol, whose values equals one if $\ell = m$ and zero otherwise. By the change of variable $\zeta = \alpha(t)x$, we are able to express the orthogonality with respect to a general w_α, which may depend on time.

From these assumptions, it is natural at this point to represent the unknown u_N through the expansion:

$$u_N(x,t) = \frac{w_\alpha(x,t)}{\sqrt{\pi}} \sum_{\ell=0}^{N} \widehat{u}_\ell(t) H_\ell\big(\alpha(t)x\big), \tag{9}$$

with its Fourier coefficients given by:

$$\widehat{u}_m(t) = \int_{\mathbb{R}} u_N(x,t)\phi_m(x,t)\, dx, \tag{10}$$

where

$$\phi_m(x,t) = \frac{\alpha(t)}{2^m\, m!} H_m\big(\alpha(t)x\big), \qquad 0 \leq m \leq N. \tag{11}$$

In the trivial case where $f = 0$, the series in (9) only contains the term corresponding to $\ell = 0$, with $\widehat{u}_0 = 2\alpha(t)$ (see (2)). In (11), the choice for ϕ_m, $0 \leq m \leq N$, is the one also suggested for the test functions in (4). Concerning the right-hand side, we have that the projection f_N of f is obtained by:

$$f_N(x,t) = \frac{w_\alpha(x,t)}{\sqrt{\pi}} \sum_{\ell=0}^{N} \widehat{f}_\ell(t) H_\ell(\alpha(t)x). \tag{12}$$

The Fourier coefficients $\widehat{f}_m(t)$ are recovered after multiplying by the test functions in (11) and integrating over $\mathbb{R}$, so obtaining:

$$\frac{\alpha(t)}{2^m\, m!} \int_{\mathbb{R}} f_N(x,t) H_m(\alpha(t)x)\, dx = \frac{1}{2^m\, m!\, \sqrt{\pi}} \sum_{\ell=0}^{N} \widehat{f}_\ell(t) \int_{\mathbb{R}} w_\alpha(x,t) H_\ell(\alpha(t)x) H_m(\alpha(t),x), \alpha(t)\, dx$$

$$= \frac{1}{2^m\, m!\, \sqrt{\pi}} \sum_{\ell=0}^{N} \widehat{f}_\ell(t) \int_{\mathbb{R}} \exp\left(-\zeta^2\right) H_\ell(\zeta) H_m(\zeta)\, d\zeta = \frac{1}{2^m\, m!\, \sqrt{\pi}} \sum_{\ell=0}^{N} \widehat{f}_\ell(t)\, 2^m m! \sqrt{\pi} \delta_{\ell,m} = \widehat{f}_m(t), \tag{13}$$

where we used the orthogonality properties of Hermite polynomials.

We can go through the computations in order to find out the scheme in finite dimension. The procedure is here omitted, though it follows from straightforward (but quite annoying) calculations. Some passages are briefly reported in the appendix . We then arrive at the following result.

Let $\widehat{u}_m(t)$ and $\widehat{f}_m(t)$ be the m-th time-dependent Fourier coefficients of u_N and f_N, respectively introduced in (9) *and* (13)*. Then, the Fourier coefficients $\widehat{u}_m(t)$, $0 \leq m \leq N$, satisfy the system of ordinary differential equations:*

$$\partial_t \widehat{u}_m(t) - \left(\frac{\alpha'(t)}{2\alpha(t)} + \alpha(t)^2\right) \widehat{u}_{m-2}(t) - (m+1)\frac{\alpha'(t)}{\alpha(t)} \widehat{u}_m(t) = \widehat{f}_m(t). \tag{14}$$

The same relation between the coefficient of the Fourier expansions is obtained from the following variational formulation of the heat equation:

$$\int_{\mathbb{R}} \Big(\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2}\Big)\phi\, dx = \int_{\mathbb{R}} f\phi\, dx \quad \forall \phi\, (\text{generic test function}), \tag{15}$$

where, for the moment, ϕ denotes a generic test function, and no integration by parts has been performed. The equivalence of the two versions is formally shown in the appendix Nevertheless, the last formulation cannot be easily

embedded in the proper functional spaces, so that an analysis of convergence for the corresponding Galerkin approximation is hard to achieve. A third version has been proposed in [24]. This is based on a rewriting of the heat equation as follows:

$$\frac{d}{dt}\int_{\mathbb{R}} u\phi\,dx - \int_{\mathbb{R}} u\left(\frac{\partial\phi}{\partial t}+\frac{\partial^2\phi}{\partial x^2}\right)dx = \int_{\mathbb{R}} f\phi\,dx \qquad \forall\phi\,(\text{generic test function}). \tag{16}$$

This leads again to the scheme (14). The reason for this last choice is due to an erroneous writing of the relation linking the Fourier coefficients of the classical Galerkin scheme (4), which led to equation (1.4) in [24]. This flaw suggested to the authors a modification of the variational formulation according to (16). After a review of the computations, we consider the alternative version (16) unnecessary. Details of the passages are provided in the appendix . It has to be remarked however that the work in [24] has been mainly focused to the study of convergence of the scheme (14), which is exactly the one we are implementing in the present paper. Therefore, the theoretical results developed in [24] are definitely interesting to us.

The exact solution (2) also satisfies the recursive relation in (14) for $f = 0$. In fact, we first recall that $\alpha(t) = 1/2\sqrt{t+1}$, which implies $\alpha'(t)/2\alpha(t) = -\alpha(t)^2$. In this way, (14) reduces to:

$$\partial_t\widehat{u}_m(t) + 2(m+1)\alpha(t)^2\widehat{u}_m(t) = \widehat{f}_m(t). \tag{17}$$

All the Hermite coefficients of u are zero with the exception of the one corresponding to $m = 0$, where we have $\hat{u}_0(t) = 2\alpha(t)$. Relation (17) is actually compatible with this choice.

Let us now suppose that the parameter α is constant. In this circumstance, always for $f = 0$, we can find for example an explicit expression for the coefficients in (14) for $m \geq 0$ (m even):

$$\widehat{u}_m(t) = \frac{\sqrt{\pi}\alpha^m}{(\frac{m}{2})!}t^{m/2}, \tag{18}$$

from which we get $\widehat{u}_0 = \sqrt{\pi}$, which corresponds to the initial datum $u(x,0) = w_\alpha(x,0)$. Indeed, by setting α constant, relation (14) becomes:

$$\partial_t\widehat{u}_m(t) - \alpha^2\widehat{u}_{m-2}(t) = 0. \tag{19}$$

Such an equation is verified by noting that:

$$\partial_t\widehat{u}_m(t) = \frac{\sqrt{\pi}\alpha^m}{(\frac{m}{2})!}\partial_t t^{m/2} = \alpha^2\frac{\sqrt{\pi}\alpha^{m-2}}{(\frac{m-2}{2})!}t^{(m-2)/2} = \alpha^2\widehat{u}_{m-2}.$$

Since, for the initial datum in (1b), only the first mode is different from zero, we have that u_0 is an even function that remains even during all the time evolution (recall that $f = 0$). Therefore, by using the Fourier coefficients in (18), we build the sum:

$$v(x,t) = w_\alpha(x)\sum_{m(even)=0}^{\infty}\alpha^m\frac{t^{\frac{m}{2}}}{(\frac{m}{2})!}H_m(\alpha x) = w_\alpha(x)\sum_{\ell=0}^{\infty}\alpha^{2\ell}\frac{t^\ell}{\ell!}H_{2\ell}(\alpha x), \tag{20}$$

where we dropped the dependence on t in w_α, since the weight no longer depends on time. We have that $v(x,0) = u(x,0)$ when $\alpha = 1/2$.

We can also say that $v(0,t) = u(0,t)$, for $0 \leq t < 1$. We prove this fact by writing the MacLaurin expansion:

$$u(0,t) = 2\alpha(t) = \frac{1}{\sqrt{t+1}} = \sum_{\ell=0}^{\infty}\frac{(-1)^\ell}{2^\ell\ell!}\Big(\prod_{k=1}^{\ell}(2i-1)\Big)t^\ell. \tag{21}$$

Now, the expansions (20) and (21) are the same, as it can be checked by expressing the value of $H_{2\ell}(0)$ through the recursion formula for Hermite polynomials. Let us observe that the above mentioned series have a convergence radius equal to 1, therefore they diverge for $t > 1$.

We are tempted to deduce that the functions v and u always coincide both in time and space. This kind of equality is not clear however, since it presumes a correct interpretation of the type of convergence of the series in (20). For any fixed x and t, we may expect point-wise convergence, but it is hard to conclude that this is going to be true in some normed functional space. In fact, we are trying to represents a Hermite function with a certain Gaussian decay,

through an expansion by other Hermite functions associated to a different Gaussian behavior. To this respect, let us note that, by choosing $\alpha(t)$ as in (2), only the first mode ($m = 0$) is activated during the whole computation, so that the numerical error exclusively depends on the time discretization procedure. On the contrary, for α fixed equal to $1/2$, all the Fourier coefficients are involved.

In [24], spectral convergence is proven for a special choice of $\alpha(t)$ (see relation (2.1) in that paper), which includes the case (2). The analysis for a more general α is at the moment unavailable. Such an extension is rather troublesome and it is due to the difficulty to find proper Sobolev type inclusions between spaces displaying different decay behaviors at infinity of the weight functions. The results of the experiments obtained by truncating the sum in (20) at a fixed integer N show an excellent agreement with the exact solution u, for $t < 1$. As t reaches the value 1, oscillations are observed. Like in the Gibbs' phenomenon, these do not diminish by increasing N (recall that the convergence radius of the involved series is equal to 1).

3. Quadrature nodes and other useful formulas

The considerations made in the previous section, rise the question of the passage from the Fourier coefficients relative to a certain choice of α to those related to a different value of such a parameter. From the practical viewpoint, we briefly provide a recipe for this kind of basis transformation. Suppose that we are given a function v expressible as a series of Hermite functions for some $\alpha > 0$:

$$v(x) = \frac{w_\alpha(x)}{\sqrt{\pi}} \sum_{\ell=0}^{\infty} \widehat{v}_\ell H_\ell\big(\alpha x\big), \tag{22}$$

with Fourier coefficients

$$\widehat{v}_m = \frac{\alpha}{2^m\, m!} \int_{\mathbb{R}} v(x) H_m(\alpha x)\, dx = \frac{1}{2^m\, m!} \int_{\mathbb{R}} v(x/\alpha) H_m(x)\, dx, \qquad m \geq 0. \tag{23}$$

We would like to express the same function as the series of Hermite functions relative to $\alpha = 1$:

$$v(x) = \frac{w_1(x)}{\sqrt{\pi}} \sum_{\ell=0}^{\infty} \widehat{V}_\ell H_\ell(x), \tag{24}$$

with Fourier coefficients

$$\widehat{V}_m = \frac{1}{2^m\, m!} \int_{\mathbb{R}} v(x) H_m(x)\, dx, \qquad m \geq 0. \tag{25}$$

We can immediately get a formula that provides the set of Fourier coefficient $\widehat{V}_m$ from the other one. It is enough to informally substitute expansion (22) in the last integral. So, we find that

$$\widehat{V}_m = \frac{1}{2^m\, m!\sqrt{\pi}} \sum_{\ell=0}^{\infty} \widehat{v}_\ell \int_{\mathbb{R}} H_\ell\big(\alpha x\big) H_m(x) w_\alpha(x)\, dx = \frac{1}{\alpha 2^m\, m!\sqrt{\pi}} \sum_{\ell=0}^{\infty} \widehat{v}_\ell \int_{\mathbb{R}} H_\ell\big(x\big) H_m\left(\frac{x}{\alpha}\right) w_1(x)\, dx \tag{26}$$

and, viceversa, substituting (24) in (23).

$$\widehat{v}_m = \frac{\alpha}{2^m\, m!\sqrt{\pi}} \sum_{\ell=0}^{\infty} \widehat{V}_\ell \int_{\mathbb{R}} H_\ell(x) H_m(\alpha x) w_1(x)\, dx, \qquad m \geq 0. \tag{27}$$

Explicit formulas are available for the computation of these integrals. The procedure looks however complex and expensive, so that we provide here below an alternative by using numerical quadrature. As usual, for a fixed integer N, we introduce the zeros of H_N. These nodes will be denoted by ξ_j^N, $j = 1, \ldots, N$. Together with the weights [11, Eq. (3.4.9)]:

$$w_j^N = \sqrt{\pi} 2^{N+1} N! \big[H_n'(\xi_j^N)\big]^{-2}, \qquad 1 \leq j \leq N, \tag{28}$$

we have the quadrature formula:

$$\int_{\mathbb{R}} p(x) dx = \sum_{j=1}^{N} p(\xi_j^N) w_j^N, \tag{29}$$

which is exact for any polynomial p of degree less than $2N$. We can pass from the point-values to the Fourier coefficients by observing that (23) can be rewritten as:

$$\widehat{v}_m = \frac{1}{2^m\, m!} \sum_{j=1}^{N} v(\xi_j^N/\alpha) H_m(\xi_j^N) w_j^N, \tag{30}$$

provided the product vH_m is a polynomial of degree less than $2N$. A direct application of the quadrature rule to (26) yields:

$$\widehat{V}_m \approx \frac{1}{\alpha 2^m\, m! \sqrt{\pi}} \sum_{\ell=0}^{N-1} \widehat{v}_\ell \sum_{j=1}^{N} H_\ell\big(\xi_j^N\big) H_m\left(\xi_j^N/\alpha\right) w_j^N, \qquad 0 \le m \le N-1, \tag{31}$$

where the sum on ℓ has been suitably truncated. The change of coefficients from a basis to another one is then obtained at a reasonable cost, although some approximation has been introduced. The writing in (31) may be represented by a linear operator associated with an $N \times N$ matrix. It must be observed that, once N has been fixed, we can get reliable approximations only if α stays within a certain range. In fact, it is necessary to have an appropriate amount of nodes in the support of the Gaussian (we practically define the "support" as the set where the Gaussian is different from zero for a given machine error precision). If α is too large, the support is narrow and all the nodes are outside the region of interest. In this case, the interpolating polynomial produces oscillations. On the contrary, for α too small, the support is wide and the nodes turn out to be concentrated at its interior, so providing a poor approximation. We will return to these issues in Section 6.

The use of quadrature formulas suggests the collocation method as a possible alternative to the Galerkin scheme (14). In order to follow this new path, it is customary to introduce the polynomial Lagrange basis:

$$l_j^N(x) = \frac{H_N(x)}{H_N'(\xi_j^N)(x-\xi_j^N)}, \ \text{ if } x \neq \xi_j^N, \qquad\qquad l_j^N(\xi_j^N) = 1,$$

so that, for any polynomial p of degree at most $N-1$ one has:

$$p(x) = \sum_{j=1}^{N} p(\xi_j^N) l_j^N(x).$$

We also recall the $N \times N$ differentiation matrices $D^{(1)} = \big\{d_{ij}^{(1)}\big\}$ and $D^{(2)} = \big\{d_{ij}^{(2)}\big\}$, with $D^{(2)} = D^{(1)}D^{(1)}$. These are recovered from the first and second derivatives of the Lagrange polynomials:

$$d_{ij}^{(1)} = \frac{dl_j^N}{dx}(\xi_i^N), \qquad\qquad d_{ij}^{(2)} = \frac{d^2 l_j^N}{dx^2}(\xi_i^N). \tag{32}$$

The explicit entries of these matrices are well known and can be found in many textbooks, e.g., [11] and [35]. We now have all the elements to construct an approximation method of collocation type. This will be developed in the next section.

4. Collocation algorithm and time discretization

We start by writing the unknown u as:

$$u(x,t) = p(x,t)\omega_\alpha(x,t), \tag{33}$$

where we recall that $\omega_\alpha(x,t) = \exp\big(-\alpha(t)^2 x^2\big)$. Note that in (33) p is not necessarily a polynomial. By substituting this expression into the heat equation we obtain:

$$\omega_\alpha \partial_t p + p\partial_t \omega_\alpha = \omega_\alpha \partial_{xx} p - 2\partial_x p\, \partial_x \omega_\alpha + p\partial_{xx}\omega_\alpha + f. \tag{34}$$

We then divide both sides of (34) by ω_α to obtain:

$$\partial_t p + 2\alpha\alpha' x^2 p = \partial_{xx} p - 4\alpha^2 x \partial_x p + 2\alpha^2\big(2\alpha^2 x^2 - 1\big)p + \frac{f}{\omega_\alpha}, \tag{35}$$

where α' denotes the derivative of α with respect to time. Finally, we look for an approximating polynomial p_N of degree less than or equal to $N-1$, after collocating the above equation at the points ξ_j^N, $1 \le j \le N$:

$$\partial_t p_N(\xi_j^N,t) + 2\alpha(t)\alpha'(t)(\xi_j^N)^2 p_N(\xi_j^N,t) = (\partial_{xx}p_N)(\xi_j^N,t) \\ - 4\alpha^2(t)\xi_j^N(\partial_x p_N)(\xi_j^N,t) + 2\alpha^2(t)\big(2(\alpha(t)\xi_j^N)^2-1\big)p_N(\xi_j^N,t) + \frac{f(\xi_j^N,t)}{\omega_\alpha(\xi_j^N,t)}. \quad (36)$$

Here, the derivatives $\partial_x p_N$ and $\partial_{xx}p_N$ can be evaluated with the help of the finite dimensional operators $D^{(1)}$ and $D^{(2)}$ defined in (32). Note that the approximate solution of the heat equation is recovered as in (33) through the expression

$$u_N(x,t) = p_N(x,t)w_\alpha(x,t). \quad (37)$$

If $\alpha(t)$ is chosen to follow the behavior of the exact solution u for $f=0$, i.e. $\alpha(t) = 1/2\sqrt{t+1}$ as in (2), the approximation is excellent. Indeed, in this simple case, p_N turns out to be a polynomial of degree zero in the variable x, i.e., $p_N(x,t)=\alpha(t)$, as it can be checked by direct substitution.

Since the exact $\alpha(t)$ is not known a priori, it is interesting to see what happens for different choices of this parameter. As in the Galerkin method, we start by considering the case of α constant in time. Thus, the scheme (36) becomes:

$$\partial_t p_N(\xi_j^N,t) = (\partial_{xx}p_N)(\xi_j^N,t) - 4\alpha^2\xi_j^N(\partial_x p_N)(\xi_j^N,t) + 2\alpha^2\big(2(\alpha\xi_j^N)^2-1\big)p_N(\xi_j^N,t) + \frac{f(\xi_j^N,t)}{\omega_\alpha(\xi_j^N)}. \quad (38)$$

The next step is to discretize in time for $t\in[0,T]$. To this end we just apply the forward Euler method, though we are conscious that more sophisticated methods are available. We denote the time step by Δt and the generic n-th point of the time grid by $t^n = n\Delta t$, $0\le n\le \mathcal{N}$, $t^0=0$, $\mathcal{N}\Delta t = T$. We obtain

$$\frac{p_N^{n+1}(\xi_j^N) - p_N^n(\xi_j^N)}{\Delta t} = (\partial_{xx}p_N^n)(\xi_j^N) - 4\alpha^2\xi_j^N(\partial_x p_N^n)(\xi_j^N) + 2\alpha^2\big(2(\alpha\xi_j^N)^2-1\big)p_N^n(\xi_j^N) + \frac{f(\xi_j^N,t^n)}{\omega_\alpha(\xi_j^N)}. \quad (39)$$

We supplement equation (39) with the initial condition p_N^0, which is derived from the initial solution $u(x,0)$.

Again we take for the moment $f=0$, in order to perform some experiments. We would like to have the chance to modify α during the evolution. The idea is to keep it constant for a certain number of iterations, and then shift to another constant value. The problem is that the piecewise constant function $\alpha(t)$ is not differentiable at the discontinuity points, so that the scheme (36) cannot be applied, as it requires the knowledge of α'. We then modify the time advancing procedure in the following fashion.

Starting from a value t^n of the time discretization grid, we keep α constantly equal to a given α_n, during m consecutive time steps, i.e. within the interval $\big[t^n, t^{n+m}\big]$. In this way, we find $p_N^n, \cdots p_N^{n+m}$ solving equation (39). Correspondingly, we also construct: $u_N^{n+k} = p_N^{n+k}\exp\big(-\alpha_n^2x^2\big)$, $0\le k\le m$. At time t^{n+m} we decide to assign a new $\alpha=\alpha_{n+m}$ (we will see later how this can be done automatically). Before continuing with the scheme (39), we need to pass from the representation of u_N^{n+m}, with the Gaussian associated to α_n, to the new one relative to α_{n+m}. In order to do this, from p_N^{n+m} we build a new polynomial such that:

$$\widetilde{p}_N(\xi_j^N) = p_N^{n+m}(\xi_j^N)\exp\big((\alpha_{n+m}^2-\alpha_n^2)(\xi_j^N)^2\big), \qquad 1\le j\le N. \quad (40)$$

This transitory $\widetilde{p}_N$ is used as a new initial guess to advance in time, until the next update of α. Of course, if $\alpha_{n+m}=\alpha_n$, this last passage is not relevant.

We now discuss the results of a series of numerical experiments, where the exact solution u is compared with its approximation u_N. The errors are evaluated in the followings norms:

$$\mathcal{N}^1 = \left(\sum_{j=0}^N |u(\xi_j^N)-u_N(\xi_j^N)|^2\right)^{1/2}, \qquad \mathcal{N}^2 = \max_{\mathbb{R}}|u-u_N|, \qquad \mathcal{N}^3 = \left(\sum_{j=0}^N |u(\xi_j^N)-u_N(\xi_j^N)|^2 w_j^N\right)^{1/2} \quad (41)$$

with $\mathcal{E}_j = |u(\xi_j^N)-u_N(\xi_j^N)|$, $1\le j\le N$. The maximum in $\mathcal{N}^2$ is computed on a very fine grid on the x-axis and w_j^N, $1\le j\le N$ are the quadrature weights introduced in (28). The time step has been chosen small enough in order

not to affect the error in space. Since the sum of the weights is equal to $\sqrt{\pi}$, the error evaluated using norm $\mathcal{N}^3$ is less than the error evaluated using norm $\mathcal{N}^1$. Therefore, with the exception of Table 1, we will not report the results related to the last norm in (41).

We continue to consider $f = 0$ and we integrate in time for $t \in [0, T]$ with $T = 1$, $\Delta t = 10^{-7}$. The first tests in Table 1, show a spectral decay of the solution when $\alpha = \alpha(t)$ is taken as in (2). Since we do not know a priori the behavior of α, the interesting part comes when we choose this parameter according to some prescribed law. In Table 2 we find the results where α has been fixed once and for all. As expected, the errors are not encouraging. An improvement is obtained by imposing that $\alpha(t)$ is piecewise constant. To this end, we subdivide the time interval in ten subintervals. In Table 3 (left panel) we have the expression of the errors when α starts from the value 0.5 and decreases up to 0.3 in 9 equal steps. In Table 3 (right panel), ten guesses of α have been made randomly in the interval $[0.3, 0.5]$. The behavior of α in all the above examples is summarized in Fig. 1.

N	$\mathcal{N}^1$	$\mathcal{N}^2$	$\mathcal{N}^3$
4	2.6171e-04	2.2846e-04	1.0907e-04
6	1.2092e-05	1.2562e-05	2.2202e-06
8	6.2025e-07	7.0862e-07	5.5019e-08
10	3.0252e-08	4.1672e-08	7.6304e-09

Table 1
Errors at time $T = 1$ (using $\Delta t = 10^{-7}$) and $\alpha(t) = 1/2\sqrt{t+1}$, $t \in [0, 1]$.

N	$\mathcal{N}^1$	$\mathcal{N}^2$
4	2.6090e-02	6.8507e-02
6	8.7970e-03	2.6723e-03
8	3.0084e-03	1.0619e-02
10	1.0421e-03	4.2718e-03

N	$\mathcal{N}^1$	$\mathcal{N}^2$
4	6.6306e-02	8.0850e-02
6	4.8930e-02	7.2890e-02
8	4.0269e-02	7.5228e-02
10	3.5001e-02	8.3357e-02

Table 2
Errors at time $T = 1$ (using $\Delta t = 10^{-7}$) and α constant in time: $\alpha = 0.5$ (left); $\alpha = 0.3$ (right).

N	$\mathcal{N}^1$	$\mathcal{N}^2$
4	1.4775e-03	2.0235e-02
6	8.6681e-05	5.2286e-03
8	1.6311e-05	1.5659e-03
10	1.3840e-06	4.6910e-04

N	$\mathcal{N}^1$	$\mathcal{N}^2$
4	9.9108e-03	1.2908e-02
6	2.6628e-03	2.2971e-03
8	9.1279e-04	1.1427e-03
10	3.0883e-04	3.9922e-04

Table 3
Errors at time $T = 1$ (using $\Delta t = 10^{-7}$) and $\alpha(t)$ varying in a step-wise fashion: α decreasing from 0.5 to 0.3 (left); α randomly chosen (right).

5. Automatic decision-making approach

In this section, we still continue to handle the case $f = 0$, whereas the non-homogeneous case will be treated in the next section. As we anticipated in the introductory section, we adopt two different ML techniques in order to predict suitable values of α, able to guarantee stability and good performance. When $f = 0$, to train the system we take a set of functions of the following type

$$g_{a,H}(x) = H \, \exp\big(-a^2 x^2\big), \tag{42}$$

where the width a of the Gaussian function is randomly chosen within the interval $[0.2, 0.6]$ and its height H is also randomly chosen within the interval $[0, 1]$. In this way we construct a total of 40 functions, that we believe to be sufficient for our goal. To each choice of the pair (a, H), we assign the two distinct sets

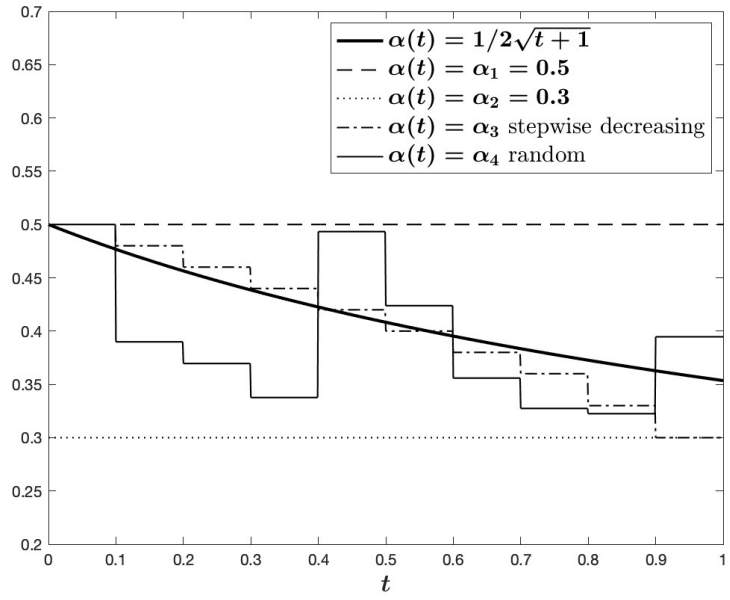

Fig. 1. Choices of α in the various experiments.

$$\mathcal{S}_{a,H}^{\mathrm{PV}} = \left\{ g_{a,H}(\xi_j^N),\ 1 \le j \le N \right\} \quad \text{and} \quad \mathcal{S}_{a,H}^{\mathrm{FC}} = \left\{ \widehat{g}_{m,a,H},\ 0 \le m \le N-1 \right\}, \tag{43}$$

where the labels PV and FC respectively stand for Point Values and Fourier Coefficients. The first set clearly contains the point-values of each Gaussian function at the Hermite nodes. The other set is represented by the first N Fourier coefficient of $g_{a,H}$. These are built as follows, for $0 \le m \le N-1$:

$$\widehat{g}_{m,a,H} = \frac{1}{2^m\, m!} \int_{\mathbb{R}} g_{m,a,H}(x) H_m(x)\, dx \approx \frac{1}{2^m\, m! \sqrt{\pi}} \sum_{j=1}^{N} g_{a,H}(\xi_j^N) \exp\left((\xi_j^N)^2\right) H_m(\xi_j^N) w_j^N \tag{44}$$

In this way, in the special situation corresponding to $a = 1$, all the quantities $\widehat{g}_{m,a,H}$ are zero for $m \ge 1$.

The first approach is suitable to the collocation setting and the second one to the Galerkin setting. However, we can easily move from one setting to the other one through the quadrature formulas of Section 3, so we can advance the numerical solution by using the scheme in (39), and indifferently make use of $\mathcal{S}_{a,H}^{\mathrm{PV}}$ or $\mathcal{S}_{a,H}^{\mathrm{FC}}$. To each set in (43), we only assign the parameter a, so that Gaussian functions displaying a different height H are considered equivalent. In this way, the algorithm is tuned on the width of the Gaussian profiles, i.e., on what determines the choice of α in the discretization of the heat equation.

For the implementation of the SVM algorithm we use the open source machine learning library: `LIBSVM` [2].We adopt a ν-SVR (ν-Support Vector Regression) approach with a radial basis function as kernel function. The system can be either trained by using the representation suggested in $\mathcal{S}_{a,H}^{\mathrm{PV}}$ or in $\mathcal{S}_{a,H}^{\mathrm{FC}}$, by varying the parameters a and H in their domains. The parameter ν allows for the control of the number of Support Vectors. The reader is addressed for more details to [2] and the references therein.

For the implementation of the deep learning algorithm, we use instead the Matlab function fitting neural network: `fitnet`, which is a feed-forward network with the tan-sigmoid transfer function in the hidden layers and linear transfer function in the output layer. Generally speaking, function fitting is the process of training a NN on a set of inputs in order to produce an associated set of target outputs. More information can be found in the Deep Learning Toolbox of Matlab (`www.mathworks.com`). Based on an "hit & trial" approach we train with the Levenberg-Marquardt algorithm [21] a NN with two hidden layers of respectively 20 and 10 neurons (see Figure 2). Note that this network has only one output neuron, because there is only one target value (i.e.: a) associated with each input vector of size $N = 10$, chosen either in $\mathcal{S}_{a,H}^{\mathrm{PV}}$ or in $\mathcal{S}_{a,H}^{\mathrm{FC}}$.

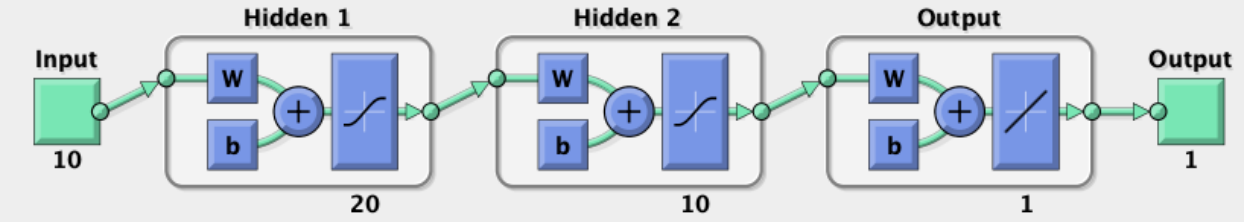

Fig. 2. Matlab neural network

We then return to the experiments of the previous section by fixing $N = 10$. We recall that the errors, when $\alpha(t)$ is taken as in (2), are given in the last row of Table 1. We would like to modify α ten times in the time interval $[0, 1]$ according to the suggestions of the machine. In the notation that follows the subscript SVM stands for Support Vector Machines and subscript DL stands for Deep Learning.

Table 4 shows the values of α obtained at various times using $\Delta t = 10^{-7}$, the two different representations of the training sets in (43) and the two different ML algorithms. The errors $\mathcal{N}^1$ and $\mathcal{N}^2$ at time $T = 1$, $\Delta t = 10^{-7}$, for various α, obtained with the two different representations of the training sets in (43) and the two different ML techniques are shown in Table 5. These results are quite satisfactory. The machine is actually capable to guess the appropriate value of α in a given situation. Of course, the errors improve if we decide to upgrade α more frequently.

Results similar to those presented in this section and the next one were also obtained by trying other choices of the network parameters, such as for example its configuration. As expected, performances may strongly depend upon the architecture and the training algorithm. According to our experience, the most critical step is to find the right number of hidden layers and their respective neurons when a NN is used.

t	$\alpha = 1/2\sqrt{t+1}$	$\alpha^{\rm FC}_{\rm SVM}$	$\alpha^{\rm FC}_{\rm DL}$	$\alpha^{\rm PV}_{\rm SVM}$	$\alpha^{\rm PV}_{\rm DL}$
0	0.5000	0.5000	0.5000	0.5000	0.5000
0.1	0.4767	0.4736	0.4474	0.4888	0.4863
0.2	0.4564	0.4549	0.3639	0.4691	0.4729
0.3	0.4385	0.4383	0.4526	0.4509	0.4613
0.4	0.4226	0.4233	0.4526	0.4340	0.4502
0.5	0.4082	0.4097	0.4526	0.4183	0.4391
0.6	0.3953	0.3972	0.4526	0.4039	0.4276
0.7	0.3835	0.3857	0.4525	0.3905	0.4159
0.8	0.3727	0.3749	0.4456	0.3781	0.4039
0.9	0.3627	0.3649	0.3734	0.3667	0.3918
1	0.3536	0.3555	0.4384	0.3560	0.3800

Table 4
Values of α at different times (using $\Delta t = 10^{-7}$) obtained with the two different representations of the training sets in (43) and two different Machine Learning techniques.

Choice of α	$\mathcal{N}^1$	$\mathcal{N}^2$
$\alpha^{\rm FC}_{\rm SVM}$	2.6985e-08	4.6072e-08
$\alpha^{\rm FC}_{\rm DL}$	2.9153e-05	5.5487e-04
$\alpha^{\rm PV}_{\rm SVM}$	4.3810e-07	5.9742e-07
$\alpha^{\rm PV}_{\rm DL}$	6.1306e-06	1.6049e-05

Table 5
Errors at time $T = 1$ (using $\Delta t = 10^{-7}$) for various α, obtained with the two different representations of the training sets in (43) and two different ML techniques.

6. The non-homogeneous case

Based on the results obtained in the previous sections, we can now discuss the case when $f \neq 0$. We set the right-hand side of (1a) in such a way that the exact solution is:

$$u(x,t) = \Big(\cos(\tfrac{1}{2}xt) + 2(t\sin(x))^2 \Big) \exp\big(-\alpha^2(t)x^2 \big) \quad \text{with} \quad \alpha(t) = \frac{\sqrt{2}}{\sqrt{3t+1}}. \tag{45}$$

With this choice α, decays from $\sqrt{2}$ to $1/\sqrt{2}$ in the time interval $[0,T]$ with $T = 1$. The behavior of u at different times is reported in Fig. 3. From a single initial bump, the solution develops by producing two asymmetric bumps. It is then hard to find the appropriate functional space with weight w_α to formalize the theoretical problem. From the practical viewpoint, we need the help of a ML algorithm.

As far as the approximation is concerned, from now on we set $N = 16$, that corresponds to polynomials of degree fifteen. All the zeroes of H_{16} are contained in the interval $[-5,5]$. Moreover we choose $\Delta t = 10^{-6}$ to discretize in time. Referring to Table 6, we soon note that the results are excellent when we take α as in (45). Nevertheless, this expression is not known a priori. Always according to Table 6, the choice of α constantly equal to $\sqrt{2}$ (recoverable from the initial datum $u(x,0)$) is a failure. We also tried to guess ten values of α randomly chosen in the interval [0.7,1.4]. This also turned out to be a bad choice. The successive step is to adopt the ML strategy discussed in the previous section. Since in the present situation the solution u is not just a Gaussian function, we need a more representative characterization of the training sets. Indeed, in order to carry out our analysis, we consider the following variant of the ML approach already experimented.

We start by defining a set of K cubic splines S_k, $1 \le k \le K$. The support of these functions is randomly chosen in the interval $[-c,c]$, where c is a parameter that must be properly adjusted. Such an interval is subdivided in $M+2$ equispaced break points, including the first endpoint at $x = -c$ and the last endpoint at $x = c$, i.e., M interior break points. The splines are required to be zero at $x = \pm c$ together with their first derivatives, and are uniquely determined by the values attained at the M internal points. Moreover, they are prolonged to zero outside the interval $[-c,c]$, in order to form a set of functions with compact support in $\mathbb{R}$. These functions are again denoted by $S_k(x)$, $1 \le k \le K$, for $x \in \mathbb{R}$, and are globally $C^1(\mathbb{R})$-regular functions. If $K \le M$, the dimension of the space generated by these splines is less than or equal to M; otherwise, if $K > M$, the splines are linearly dependent.

A given set of K splines is obtained by assigning the values at the break points in a random way. These values must be positive and less than or equal to a given upper bound $\mathcal{M}$, which is another parameter that must properly be set in the algorithm. For our experiments, we choose $K = 40$, $M = 5$, $c = 4.5$, $\mathcal{M} = 1$.

Other sets of K functions can be employed without altering the nature and the main features of the ML algorithm, e.g., trigonometric functions. However, it is preferable that this family is not constituted by polynomials, since we want to introduce elements in the training set that are external to the space of the classical Hermite functions.

For each cubic spline S_k and a given value of α, we look for the interpolating polynomial of degree less than or equal to $N-1$ such that:

$$\widetilde{p}_{N,k}(\xi_j^N) = S_k(\xi_j^N) \exp\big((\alpha\xi_j^N)^2\big), \qquad 1 \le j \le N. \tag{46}$$

Afterwards, we compute the following quantity on a finer grid:

$$\Gamma_{N,k}(\alpha) = \max_{x\in\mathbb{R}} \Big| S_k(x) - \widetilde{p}_{N,k}(x)\exp(-\alpha^2x^2) \Big|. \tag{47}$$

Assuming that the parameter α is varying within a given interval I, we finally determine:

$$\Gamma_{N,k}(\alpha_k) = \min_{\alpha\in I} \Gamma_{N,k}(\alpha). \tag{48}$$

The Hermite function corresponding to α_k,

$$g_k(x) = \widetilde{p}_{N,k}(x)\exp(-\alpha_k^2x^2), \tag{49}$$

is called the *minimizing Hermite function* associated to the spline S_k. In this way, for any S_k, we obtains pairs like (g_k, α_k), The above procedure extends to larger functional sets as the ones already considered in the previous section. In particular, if we replace S_k by a Gaussian function of the type $g_{a,H}$ like in (33), the *minimax*-like procedure just introduced would suggest taking $\alpha = a$ and $\widetilde{p}_{N,k}$ constantly equal to H.

An essay of this construction is visible in Fig. 4. On the left, three examples of randomly generated splines are displayed together with their corresponding minimizing Hermite functions. On the right, we show the semi-log plot of $\Gamma_{N,k}(\alpha)$ as a function of α. It is clear that there is a minimum value $\alpha = \alpha_k$, which is the one actually used to detect the function in (49). To obtain these graphs, we assumed that the values attained by the splines were included in the interval $[0, \mathcal{M}] = [0, 1]$. Moreover, we took $I = [0.5, 1.5]$.

Similarly to (43), we can finally define the two training sets as follows. The minimizing Hermite functions can be represented through the values attained at the zeroes of the Hermite polynomial H_N. In this way, we can build the first set of training functions:

$$\mathcal{S}_k^{\rm PV} = \Big\{ g_k(\xi_j^N) = S_k(\xi_j^N),\ \ 1 \le j \le N \Big\}, \qquad\qquad 1 \le k \le K. \tag{50}$$

Any $\mathcal{S}_k^{\rm PV}$ comes with its own α_k, obtained through (48). The network is actually trained with the help of the values associating S_k to α_k. In alternative, it is possible to consider the Fourier coefficients computed through (44). In this case, we define the elements of a new training set as:

$$\mathcal{S}_k^{\rm FC} = \Big\{ \widehat{g}_{m,k},\ \ 0 \le m \le N-1 \Big\}, \qquad\qquad 1 \le k \le K. \tag{51}$$

It is worthwhile to anticipate that the experiments are very sensitive to the choice of the values of the various parameters, which are N, K, M, $\mathcal{M}$, c, I. In Table 6, we provide the errors at $T = 1$, using $\Delta t = 10^{-6}$, for different choices of α. Things go smooth if we take $\alpha = \alpha(t)$ as in (45). However this behavior is not available, with the exception of the initial value where $\alpha(0) = \sqrt{2}$. On the other hand, if we fix α to be constantly equal to $\sqrt{2}$ the results are a disaster. Divergence is also observed when α is randomly chosen in the interval $[0.5, 1.5]$ with changes at times $t^\ell = 0.1\ell$, $1 \le \ell \le 9$.

With the automatic strategy, we again start from an initial value of α equal to $\sqrt{2}$, but, then, we follow the suggestion of the machine. We still use the SVM algorithm and the deep learning algorithm described in the previous section. In particular the results of the following experiments are obtained using a feed-forward Matlab network with two layers of five neurons each. As done in the previous section, the update of α is made 9 times in the time interval $(0, 1)$. The final results are rather good. The parameter changes drastically at time $t = 0.1$, and then, following small variations, approximately stabilizes around the value $\alpha \approx 0.8$ (we do not show this graph). This is why, in the last row of Table 6 we report the errors corresponding to the case where α has been kept fixed for all the computation. Without the help of the learning procedure, we would not be able to guess that $\alpha = .8$ is a good choice.

Similar observations were made in [9] regarding the approximation of the 1D-1D Vlasov equation through a periodic Fourier expansion in the space variable x and Hermite functions in the velocity variable v. In that paper, for N relatively small, it is numerically proven that the impact due to a fixed choice of α is relevant. The results show that the best performance is obtained for unexpected guesses of α, whereas the natural choice associated with the initial datum leads to an embarrassing instability.

One may be surprised from observing that the figures of Table 6 look very similar. We can try to give an explanation of this fact. We realized in these numerical investigations, that the spectral collocation method converges independently of the setting up of the functional spaces if α remains within a certain range, see the end of Section 2. Proving convergence for a weight w_α that does not match the decay of the solution u is a theoretical difficulty that has little influence on the practical outcome. Changing α during the time-advancing process alters the functional "habitat" of the approximated solution, but not its capacity to produce reliable results. Therefore, at least in the framework of the linear problem studied so far, the guided choice of α is primarily useful to achieve stability. This is a nontrivial requisite, since the brutal choice $\alpha = \sqrt{2}$, which agrees with the initial datum, is far from being effective. This phenomenon is less evident if we examine the tables relative to the case $f = 0$ (section 5), where the errors are indeed affected by the modification of α. Those figures were obtained using smaller values of N. In our opinion, even if the proposed values of α do allow stability, there are still no sufficient nodes $\xi_j^N, 1 \le j \le N$, in the support of the Gaussians, to guarantee an extremely accurate approximation. This fact agrees with the observations made in [41] and recalled here in the introductory section. We also made similar observations at the end of Section 3. This remark stresses how critical is the set up of all the parameters involved in such a kind of computations.

What makes the above discussion worthwhile of attention is the fact that to find an optimal value of the parameter α might not be so crucial for improving the accuracy of the approximation, although it could still be important for the stability. When N is sufficiently large and α is not exaggeratedly out of range, the discretization method provides

Choice of α	$\mathcal{N}^1$	$\mathcal{N}^2$
α as in (45)	5.8572e-04	3.2432e-04
$\alpha = \sqrt{2}$	does not converge	does not converge
α random	does not converge	does not converge
$\alpha^{\rm FC}_{\rm SVM}$	7.5312e-04	4.4422e-04
$\alpha^{\rm FC}_{\rm DL}$	7.5340e-04	4.5743e-04
$\alpha^{\rm PV}_{\rm SVM}$	7.5310e-04	4.4997e-04
$\alpha^{\rm PV}_{\rm DL}$	7.5330e-04	4.4862e-04
α=0.8	7.5311e-04	4.4292e-04

Table 6
Errors at time $T = 1$ (using $\Delta t = 10^{-6}$) for different values of α obtained with two different representations of the training sets ((50) and (51)) and two different ML techniques.

results that are very mildly dependent on the choice of α. Of course, there are borderline situations, that can be frequent in applications, where N cannot be taken too large. In those circumstances, an automatic technique to detect α comes in handy. It has however to be said that, in general, the difficulty of finding reliable training sets is quite a crucial issue in the organization of ML strategy. This may actually present a drawback in some circumstances.

We end this section with an observation. Why did we use a ML technique instead of a plain Least Square approach? By assuming that the couples $\left(\{V_{k,j}\}_{1\le j\le N}, \alpha_k\right)$ are elements of our training set for $1 \le k \le K$, the standard Least Square method requires first of all the introduction of some weights $\bar{W} = \{W_j\}_{1\le j\le N}$. By defining the quadratic functional

$$F(\bar{W}) = \sum_{k=1}^{K} \left(\sum_{j=1}^{N} V_{k,j} W_j \ - \ \alpha_k \right)^2 , \tag{52}$$

we would like to find the set of weights $\bar{W}_{\min} = \{W_{\min,j}\}_{1\le j\le N}$ in such a way that

$$F(\bar{W}_{\min}) = \min_{\bar{W}} F(\bar{W}). \tag{53}$$

Once this has been done, let $\{V^\star_j\}_{1\le j\le N}$ be a new set of entries that correspond to some smooth function having compact support. We then assign a value $\alpha^\star$, according to:

$$\alpha^\star = \sum_{j=1}^{N} V^\star_j \ W_{\min,j}. \tag{54}$$

Unfortunately, preliminary numerical results showed that such an estimate of $\alpha^\star$ is not reliable. The reasons for the mismatch are the following. If $K \ll N$, there are not enough functions in the training set to appropriately describe the new entry. If $K > M$, there are too few break points to describe the spline set. Thus, the splines are linearly dependent and the minimization process to determine $\bar{W}$ does not give satisfactory results, since the determinant of the Linear Regression matrix associated with the minimization problem (53) turns out to be zero. If $M \approx N$, there are too many break points. For this reason, the splines may present oscillations which are not properly caught by the corresponding minimizing Hermite function (the situation is even worse if $M > N$). In all cases, the learning procedure may be considered unsatisfactory. Instead, a nonlinear ML approach allows us to introduce a larger number of weights in the system. As a consequence, we can play with more functions in the training set, and this is true also when $K > M$, i.e., when the splines start to be linearly dependent. The just mentioned situation is rather different from those emerging, for instance, from problems in Image Recognition. There, the number K of pictures in the training set is much less that M, which is the number of pixels needed to represent a single image.

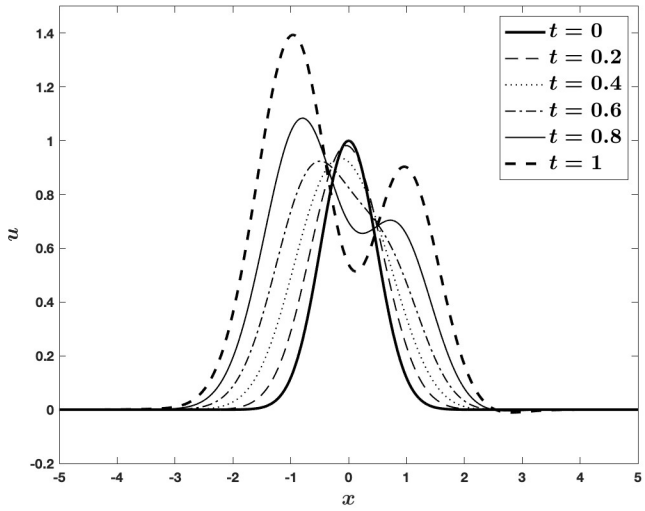

Fig. 3. The non homogeneous case: the solution u given in (45) at different times.

7. Conclusions and future work

In this work, we proposed an automatic decision-making system for the determination of the scaling factor α of the weight function in the Hermite spectral method for a PDEs on unbounded domains. An appropriate value of such parameter is crucial when using this kind of approximations since a bad choice may result in instabilities or impractical costs of implementation. We employed ML techniques based on either deep NN or SVM, in order to predict α in automatic way, in the case of a one-dimensional time-dependent equation. After the training of the machine, the algorithm advances in time by updating periodically α according to a procedure that extracts the value of the parameter in agreement to the behavior manifested in phase of evolution by the solution itself. The numerical investigations carried out show that the algorithm is able to determine α and maintain the stability, so improving when possible the accuracy of the outcome. As a proof of concept and for the exposition sake, we applied this approach to the 1D heat equation, but we are confident that applications to non-linear multi-dimensional problems can turn out to be successful, in particular in the domain of plasma physics.

In the case of the 1D-1V Vlasov equation, the best known algorithm to select the shifting and scaling factors relies on a physically-based criterion, by following the average velocity and temperature of each plasma species, i.e., the first and second moments of the plasma phase space density [5]. This time-dependent adaptive strategy has been proved to preserve the conservation laws of total mass, momentum and energy of the non-adaptive approach. All these considerations dictate the direction of our future work, i.e., the design of ML strategies for the automatic determination of both α and β in the spectral approximations of the Vlasov equation. This fulfillment requires additional efforts, due to the fact that $\alpha = \alpha(t, \boldsymbol{x})$ and $\beta = \beta(t, \boldsymbol{x})$ now also depend on the location of the particles. A straightforward implication is that the design of the training set becomes much more complicated.

Acknowledgments

The Authors are grateful to Dr. G. L. Delzanno (LANL) and Prof. C. Pagliantini for many fruitful discussions and suggestions. The Authors are affiliated to GNCS-INdAM (Italy). The third author was supported by the LDRD program of Los Alamos National Laboratory under project number 20170207ER. Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of U.S. Department of Energy (Contract No. 89233218CNA000001).

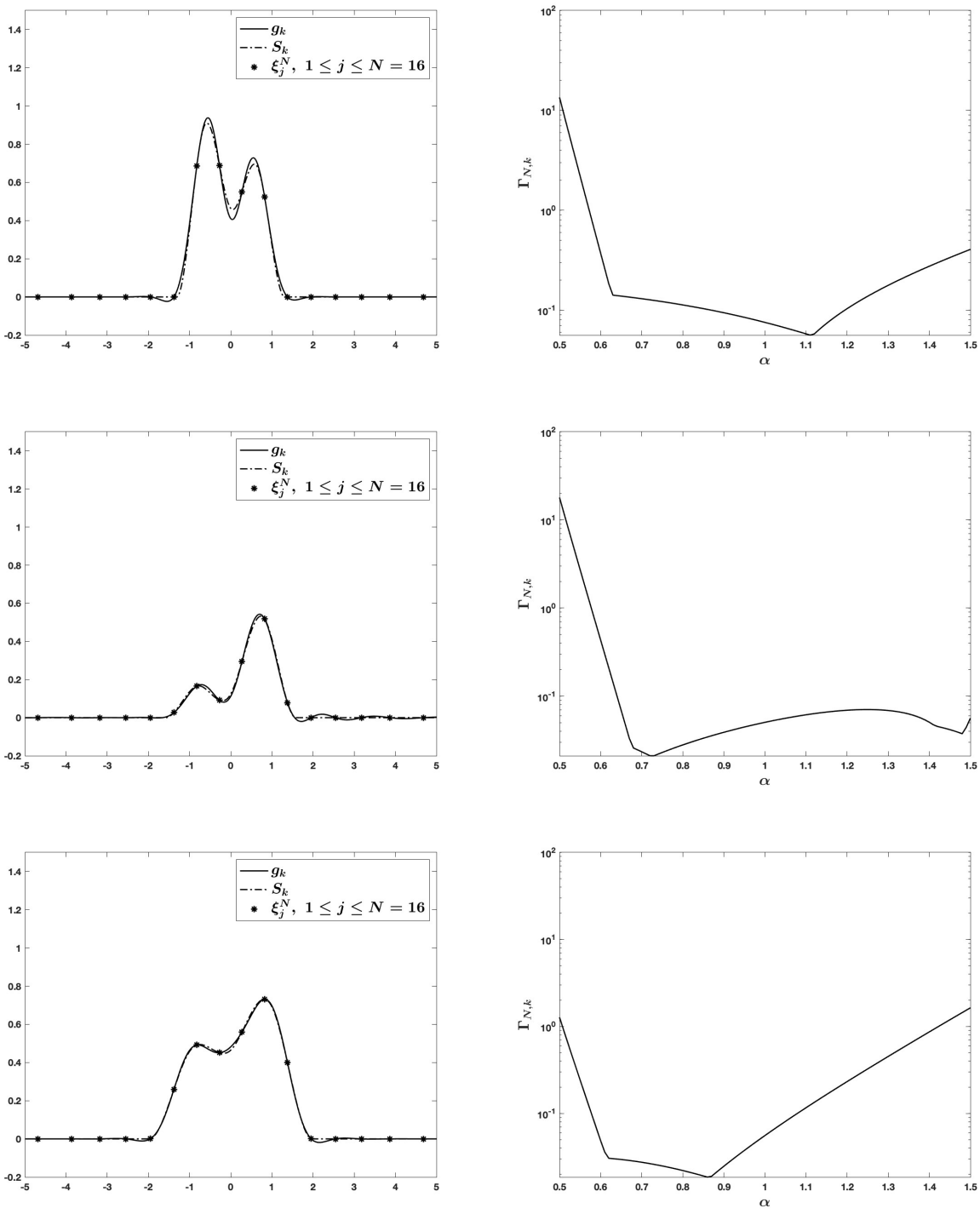

Fig. 4. Plots (left) of some randomly chosen splines overlapped to the corresponding minimizing Hermite function (49) for $x \in [-5, 5]$. Plots (right) in semi-log scale of $\Gamma_{N,k}(\alpha)$ as a function of α. The minimizing values α_k are respectively 1.11 (top), 0.73 (middle), 0.86 (bottom).

References

[1] E. Camporeale, G. L. Delzanno, G. Lapenta, and W. Daughton. New approach for the study of linear Vlasov stability of inhomogeneous systems. *Physics of Plasmas*, 13(9):092110, 2006.

[2] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:1–27, 2011. Software available at *http://www.csie.ntu.edu.tw/ cjlin/libsvm*.

[3] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2:303–314, 1989.

[4] G. L. Delzanno. Multi-dimensional, fully-implicit, spectral method for the Vlasov-Maxwell equations with exact conservation laws in discrete form. *Journal of Computational Physics*, 301:338–356, 2015.

[5] G. L. Delzanno, G. Manzini, C. Pagliantini, and S. Markidis. Physics-based adaptivity of a spectral method for the Vlasov-Poisson equations based on the asymmetrically-weighted Hermite expansion in velocity space. Tech. Report (unpublished) LA-UR-19-29686, Los Alamos National Laboratory, Los Alamos, New Mexico, USA, 9 2019.

[6] G. L. Delzanno and V. Roytershtein. Spectral approach to plasma kinetic simulation based on Hermite decomposition in velocity space. *Frontiers in Astronomy and Space Sciences*, 5, 2018. (LA-UR-18-23813, DOI:10.3389/fspas.2018.00027).

[7] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. *Advances in Neural Information Processing Systems*, 9:(7pp), 11 2003.

[8] L. Fatone, D. Funaro, and G. Manzini. Arbitrary-order time-accurate semi-Lagrangian spectral approximations of the Vlasov-Poisson system. *J. Comput. Phys.*, 384:349–375, 2019.

[9] L. Fatone, D. Funaro, and G. Manzini. A semi-Lagrangian spectral method for the Vlasov-Poisson system based on Fourier, Legendre and Hermite polynomials. *Comm. Appl. Math. Comput.*, 1:333–360, 2019.

[10] L. Fatone, D. Funaro, and G. Manzini. On the use of Hermite functions for the Vlasov-Poisson system. In S. J. Sherwin, D. Moxey, J. Peirã, P. E.; Vincent, and C. Schwab, editors, *Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2018*, volume 134 of *Lecture Notes in Computational Science and Engineering*, pages 143–153, London, UK, July, 9-13 2020. ICOSAHOM, LNCSE. Selected Papers from the ICOSAHOM Conference.

[11] D. Funaro. *Polynomial approximation of differential equations*, volume 8. Springer Science & Business Media, 1992.

[12] D. Funaro and O. Kavian. Approximation of some diffusion evolution equations in unbounded domains by Hermite functions. *Mathematics of Computation*, 57(196):597–619, 1991.

[13] D. Funaro and G. Manzini. Stability and conservation properties of Hermite-based approximations of the Vlasov-Poisson system, 2020. submitted to J. Sci. Comput.

[14] D. Gottlieb and S. A. Orszag. *Numerical Analysis of Spectral Methods (Theory and Applications) – Front Matter*, volume 10.1137/1.9781611970425. SIAM, 1977.

[15] H. Grad. On the kinetic theory of rarefied gases. *Communications on Pure and Applied Mathematics*, 2(4):331–407, 1949.

[16] B.-Y. Guo. Error estimation of Hermite spectral method for nonlinear partial differential equations. *Mathematics of Computation*, 68(227):1067–1078, 1999.

[17] I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large VC-dimension classifiers. In J.D. Cowan S.J. Hanson and C.L. Giles, editors, *Advances in Neural Information Processing Systems*, pages 147–155. Morgan Kaufmann Publishers, 1993.

[18] T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36:1171–1220, 6 2008.

[19] X. Jin, S. Cai, H. Li, and G. E. Karniadakis. NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *J. Comput. Phys.*, pages 109951–, 11 2020.

[20] G. E. Karniadakis and J. Hesthaven. Machine learning for physical systems (special issue). J. Comput. Phys., 2019. https://www.sciencedirect.com/journal/journal-of-computational-physics/special-issue/10QSWHT6XHF.

[21] C. T. Kelley. *Iterative Methods for Optimization*, volume 10.1137/1.9781611970920. SIAM Publications, 1999.

[22] E. Kharazmi, , Z. Zhang, and G .E. Karniadakis. Variational physics-informed neural networks for solving partial differential equations, 2019. arXiv:1912.00873.

[23] X. Luo, S.-T. Yau, and S. S.-T. Yau. Time-dependent Hermite-Galerkin spectral method and its applications. *Appl. Math. Comput.*, 264(C):378–391, 2015.

[24] H. Ma, W. Sun, and T. Tang. Hermite spectral methods with a time-dependent scaling for parabolic equations in unbounded domains. *SIAM J. Numer. Anal.*, 43(1):58–75, 2005.

[25] G. Manzini, G. L. Delzanno, J. Vencels, and S. Markidis. A Legendre-Fourier spectral method with exact conservation laws for the Vlasov-Poisson system. *Journal of Computational Physics*, 317:82–107, 2016.

[26] G. Manzini, D. Funaro, and G. L. Delzanno. Convergence of spectral discretizations of the Vlasov-Poisson system. *SIAM J. Numer. Anal.*, 55(5):2312–2335, 2017.

[27] T. Nguyen-Thien and T. Tran-Cong. Approximation of functions and their derivatives: A neural network implementation with applications. *Applied Mathematical Modelling*, 23:687–704, 1999.

[28] G. Pang, M. D'Elia, M. Parks, and G. E. Karniadakis. nPINNs: Nonlocal physics-informed neural networks for a parametrized nonlocal universal Laplacian operator. Algorithms and applications. *J. Comput. Phys.*, pages 109760–, 8 2020.

[29] A. Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195, 1 1999.

[30] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-Informed Neural Networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, pages S0021999118307125–, 11 2018.

[31] P. Ramuhalli, L. Udpa, and S. S. Udpa. Finite-element neural networks for solving differential equations. *IEEE Transactions on Neural Networks*, 16:1381–1392, 2005.

[32] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks ICANN-96*, volume 1112 of *Lecture Notes in Computer Science*, pages 47–52, Berlin, 1996. Springer-Verlag.

[33] B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization and beyond*. Adaptive computation and machine learning. MIT Press, 2009.

[34] J. W. Schumer and J. P. Holloway. Vlasov simulations using velocity-scaled Hermite representations. *J. Comput. Phys.*, 144(2):626–661, 1998.

[35] J. Shen, T. Tang, and L.-L. Wang. *Spectral Methods: Algorithms, Analysis and Applications*. Springer Publishing Company, Incorporated, first edition, 2011.

[36] Y. Shin, J. Darbon, and G. E. Karniadakis. On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes. *Communications in Computational Physics*, 28(5):2042–2074, 2020.

[37] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, pages 1339–1364, 8 2018.

[38] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 08 2004.

[39] M. Stoffel, F. Bamer, and B. Markert. Artificial neural networks and intelligent finite elements in non-linear structural mechanics. *Thin-Walled Structures*, 131:102–106, 2018.

[40] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9:293–300, 06 1999.

[41] T. Tang. The Hermite spectral method for Gaussian-type functions. *SIAM J. Sci. Comput.*, 14(3):594–606, 1993.

[42] V Vapnik. The support vector method of function estimation. In J. A. K. Suykens and J. Vandewalle, editors, *Nonlinear Modeling. Advanced Black-Box Techniques*, chapter 3, pages 55–85. Springer, 1998.

[43] V. Vapnik and A. Y. Chervonenkis. On a class of perceptrons. *Automation and Remote Control*, 25(1):103–109, 1964.

[44] V. Vapnik and S. Kotz. *Estimation of Dependences Based on Empirical Data*. Springer Series in Statistics. Springer-Verlag, Berlin, New York, first edition, 1982.

[45] V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780, 1963.

[46] V. N. Vapnik. *The Nature of Statistical Learning Theory*, volume 10.1007/978-1-4757-3264-1. Springer, New York, second edition, 2000.

[47] J. Vencels, G. L. Delzanno, G. Manzini, S. Markidis, I. Bo Peng, and V. Roytershteyn. SpectralPlasmaSolver: a spectral code for multiscale simulations of collisionless, magnetized plasmas. *J. Phys. Conf. Series*, 719(1):012022, 2016.

[48] T.-J. Wang, C. Zhang, and Q. Zhang. Mixed spectral method for heat transfer using generalised Hermite functions and Legendre polynomials. *East Asian Journal on Applied Mathematics*, 6:448–465, 11 2016.

[49] Y. Yadav, A. Yadav, and M. Kumar. *An Introduction to Neural Network Methods for Differential Equations*. SpringerBriefs in Applied Sciences and Technology / SpringerBriefs in Computational Intelligence. Springer Netherlands, first edition, 2015.

Appendix

We begin by collecting some basic relations concerning Hermite polynomials. We denote with a prime the derivative of a given Hermite polynomial with respect to the argument ζ. Thus, the first and the second derivatives of the ℓ-th Hermite polynomial, for $\ell \geq 2$, are given by:

$$H'_\ell(\zeta) = 2\ell H_{\ell-1}(\zeta), \qquad H''_\ell(\zeta) = 4\ell(\ell-1)H_{\ell-2}(\zeta). \tag{55}$$

In addition, we have: $H'_0(\zeta) = 0$, $H'_1(\zeta) = 2$, $H''_0(\zeta) = 0$, $H''_1(\zeta) = 0$, so that (55) formally holds for any $\ell \geq 0$. Useful recursive relations are:

$$2\zeta H'_\ell(\zeta) = H''_\ell(\zeta) + 2\ell H_\ell(\zeta) = \begin{cases} 2\ell H_\ell(\zeta), & \ell < 2, \\ 4\ell(\ell-1)H_{\ell-2}(\zeta) + 2\ell H_\ell(\zeta), & \ell \geq 2. \end{cases} \tag{56}$$

$$\begin{aligned} \zeta H_\ell(\zeta) &= \frac{1}{4(\ell+1)}\, 2\zeta H'_{\ell+1}(\zeta) = \frac{1}{4(\ell+1)}\Big[4(\ell+1)\ell H_{\ell-1}(\zeta) + 2(\ell+1)H_{\ell+1}(\zeta)\Big] \\ &= \ell H_{\ell-1}(\zeta) + \frac{1}{2}H_{\ell+1}(\zeta), \qquad \ell \geq 1. \end{aligned} \tag{57}$$

For completeness, we note that for $\ell = 0$ it holds $\zeta H_0(\zeta) = H_1(\zeta)/2$.

We now go through the computations relative to the three formulations (5), (15), (16). We start with the second one.

We recall that α depends on t and we denote by α' its derivative. Concerning Hermite polynomials, the prime will continue to denote the derivative with respect to ζ. We substitute the definitions (11), (9) of, respectively, ϕ_m, u_N, and we split the integral in three parts that will be computed separately:

$$\begin{aligned} \int_{\mathbb{R}} \partial_t u_N\, \phi_m\, dx &= \int_{\mathbb{R}} \frac{\partial_t w_\alpha(x,t)}{\sqrt{\pi}} \left[\sum_{\ell=0}^{N} \widehat{u}_\ell(t) H_\ell\big(\alpha x\big)\right] \frac{\alpha}{2^m\, m!} H_m\big(\alpha x\big)\, dx \\ &\quad + \int_{\mathbb{R}} \frac{w_\alpha(x,t)}{\sqrt{\pi}} \sum_{\ell=0}^{N} \left[\Big(\partial_t \widehat{u}_\ell(t)\Big) H_\ell\big(\alpha x\big) + \widehat{u}_\ell(t)\, \alpha' x H'_\ell\big(\alpha x\big)\right] \frac{\alpha}{2^m\, m!} H_m\big(\alpha x\big)\, dx \\ &= \frac{\alpha}{2^m\, m! \sqrt{\pi}} (-2\alpha\alpha') \sum_{\ell=0}^{N} \widehat{u}_\ell(t) \int_{\mathbb{R}} x^2 w_\alpha(x,t) H_\ell(\alpha x) H_m(\alpha x)\, dx \\ &\quad + \frac{1}{2^m\, m!\, \sqrt{\pi}} \sum_{\ell=0}^{N} \Big[\partial_t \widehat{u}_\ell(t) \int_{\mathbb{R}} w_\alpha(x,t) H_\ell\big(\alpha x\big) H_m\big(\alpha x\big)\, \alpha\, dx \\ &\quad + \widehat{u}_\ell \alpha' \int_{\mathbb{R}} w_\alpha(x,t)\, x H'_\ell\big(\alpha x\big)\, H_m\big(\alpha x\big)\, \alpha\, dx\Big] = \textbf{(I)} + \textbf{(II)} + \textbf{(III)}. \end{aligned} \tag{58}$$

To compute **(I)**, we substitute $\zeta = \alpha x$ and use the orthogonality properties of the Hermite polynomials to find that:

$$\begin{aligned} \textbf{(I)} &= \frac{1}{2^m\, m! \sqrt{\pi}} \frac{-2\alpha'}{\alpha} \sum_{\ell=0}^{N} \widehat{u}_\ell(t) \int_{\mathbb{R}} (\alpha x)^2 \exp\big(-(\alpha x)^2\big) H_\ell(\alpha x) H_m(\alpha x) \alpha\, dx \\ &= -\frac{\alpha'}{\alpha}\Big[(2m+1)\, \widehat{u}_m(t) + 2(m+2)(m+1)\, \widehat{u}_{m+2}(t) + \frac{1}{2}\widehat{u}_{m-2}(t)\Big]. \end{aligned}$$

To compute the term **(II)**, we substitute $\zeta = \alpha x$ and use the orthogonality properties:

$$\textbf{(II)} = \frac{1}{2^m\, m!\, \sqrt{\pi}} \sum_{\ell=0}^{N} \partial_t \widehat{u}_\ell(t) \int_{\mathbb{R}} \exp(-\zeta^2) H_\ell(\zeta) H_m(\zeta)\, d\zeta = \partial_t \widehat{u}_m(t).$$

To compute term **(III)** we use (55) and the orthogonality of Hermite polynomials:

$$\textbf{(III)} = \frac{1}{2^m\, m!\, \sqrt{\pi}} \sum_{\ell=0}^{N} \widehat{u}_\ell(t) \frac{\alpha'}{\alpha} \int_{\mathbb{R}} \exp(-\zeta^2)\, \zeta H'_\ell(\zeta) H_m(\zeta)\, d\zeta$$

$$= \frac{1}{2^m\, m!} \sum_{\ell=0}^{N} \widehat{u}_\ell(t) \frac{\alpha'}{\alpha} \Big[2\ell m\, 2^{m-1} (m-1)!\, \delta_{\ell-1,m-1} + \ell\, 2^{m+1} (m+1)!\, \delta_{\ell-1,m+1} \Big]$$

$$= \frac{\alpha'}{\alpha} \Big[m\widehat{u}_m(t) + 2(m+2)(m+1)\widehat{u}_{m+2}(t) \Big].$$

By collecting the results for **(I)**, **(II)**, and **(III)**, we find out that:

$$\textbf{(I)} + \textbf{(II)} + \textbf{(III)} = \partial_t \widehat{u}_m(t) + \frac{\alpha'}{\alpha} \Big[-(m+1)\widehat{u}_m(t) - \frac{1}{2}\widehat{u}_{m-2}(t) \Big]. \tag{59}$$

In a similar fashion, we split the integral of the second derivative of u_N against the test function into three parts:

$$\int_{\mathbb{R}} \partial_{xx} u_N \phi_m\, dx = \int_{\mathbb{R}} \partial_{xx} w_\alpha(x,t) \Big[\frac{1}{\sqrt{\pi}} \sum_{\ell=0}^{N} \widehat{u}_\ell(t) H_\ell(\alpha x) \Big] \Big[\frac{\alpha}{2^m\, m!} H_m\big(\alpha x\big) \Big]\, dx$$

$$+ \frac{1}{\sqrt{\pi}} \int_{\mathbb{R}} \Big[2\partial_x w_\alpha(x,t) \sum_{\ell=0}^{N} \widehat{u}_\ell(t) \partial_x H_\ell(\alpha x) \Big] \Big[\frac{\alpha}{2^m\, m!} H_m\big(\alpha x\big) \Big]\, dx$$

$$+ \frac{1}{\sqrt{\pi}} \frac{\alpha}{2^m\, m!} \int_{\mathbb{R}} \Big[w_\alpha(x,t) \sum_{\ell=0}^{N} \widehat{u}_\ell(t) \partial_{xx} H_\ell(\alpha x) \Big] \Big[H_m\big(\alpha x\big) \Big]\, dx$$

$$= \textbf{(A)} + \textbf{(B)} + \textbf{(C)}.$$

Using the property of the Hermite polynomials, we evaluate these terms as follows:

$$\textbf{(A)} = 2\alpha^2 \Big[2m\widehat{u}_m(t) + 2(m+2)(m+1)\widehat{u}_{m+2}(t) + \frac{1}{2}\widehat{u}_{m-2}(t) \Big]$$

$$\textbf{(B)} = -4\alpha^2 \Big[m\widehat{u}_m(t) + 2(m+2)(m+1)\widehat{u}_{m+2}(t) \Big]$$

$$\textbf{(C)} = 4(m+2)(m+1)\alpha^2 \widehat{u}_{m+2}(t).$$

Putting all together we arrive at:

$$\textbf{(A)} + \textbf{(B)} + \textbf{(C)} = 2\alpha^2 \Big[2m\widehat{u}_m(t) + 2(m+2)(m+1)\widehat{u}_{m+2}(t) + \frac{1}{2}\widehat{u}_{m-2}(t) \Big]$$

$$- 4\alpha^2 \Big[m\widehat{u}_m(t) + 2(m+2)(m+1)\widehat{u}_{m+2}(t) \Big]$$

$$+ 4(m+2)(m+1)\alpha^2 \widehat{u}_{m+2}(t) = \alpha^2 \widehat{u}_{m-2}(t). \tag{60}$$

By equating (59) and (60) we finally obtain the scheme (14).

We then examine the scheme originating from (5). We must compute:

$$\int_{\mathbb{R}} \frac{\partial u_N}{\partial x} \frac{\partial \phi_m}{\partial x}\, dx = \frac{1}{\sqrt{\pi}} \frac{\alpha}{2^m\, m!} \sum_{\ell=0}^{N} \widehat{u}_\ell(t) \int_{\mathbb{R}} \partial_x \Big(w_\alpha(x,t) H_\ell(\alpha x) \Big)\, \partial_x H_m(\alpha x)\, dx$$

$$= \frac{1}{\sqrt{\pi}} \frac{\alpha}{2^m\, m!} \sum_{\ell=0}^{N} \widehat{u}_\ell(t) \Big[\int_{\mathbb{R}} \big(\partial_x w_\alpha(x,t) \big) H_\ell(\alpha x)\, \partial_x H_m(\alpha x)\, dx$$

$$+ \int_{\mathbb{R}} w_\alpha(x,t) \partial_x H_\ell\big(\alpha x\big)\, \partial_x H_m\big(\alpha x\big)\, dx \Big]$$

$$= \frac{1}{\sqrt{\pi}}\,\frac{\alpha}{2^m\,m!}\sum_{\ell=0}^{N}\widehat{u}_\ell(t)\Big[-2\alpha\int_{\mathbb{R}}(\alpha x)w_\alpha(x,t)H_\ell(\alpha x)\,2mH_{m-1}(\alpha x)\,\alpha\,dx$$

$$+\alpha\int_{\mathbb{R}}w_\alpha(x,t)H'_\ell(\alpha x)\,H'_m(\alpha x)\,\alpha\,dx\Big]$$

$$= \frac{1}{\sqrt{\pi}}\,\frac{\alpha^2}{2^m\,m!}\sum_{\ell=0}^{N}\widehat{u}_\ell(t)\Big[-4m\int_{\mathbb{R}}\zeta H_\ell(\zeta)H_{m-1}(\zeta)e^{-\zeta^2}\,d\zeta$$

$$+\int_{\mathbb{R}}\exp\left(-\zeta^2\right)H'_\ell(\zeta)\,H'_m(\zeta)e^{-\zeta^2}\,d\zeta\Big]$$

$$= \alpha^2\Big(-2m\,\widehat{u}_m(t)-\,\widehat{u}_{m-2}(t)\Big)+\alpha^2\Big(2m\,\widehat{u}_m(t)\Big) = -\alpha^2\widehat{u}_{m-2}(t). \tag{61}$$

By changing the sign of the last term in (61) we exactly get the same result in (60) which brings again to the scheme (14).

We finally examine the scheme originating from (16). The first integral is clearly equal to $\widehat{u}_m(t)$. Successively, we evaluate:

$$\int_{\mathbb{R}}u_N\,\frac{\partial^2\phi_m}{\partial x^2}\,dx = \alpha^2\int_{\mathbb{R}}u_N\,\phi_{m-2}\,dx$$

$$= \alpha^2\frac{1}{\sqrt{\pi}}\,\frac{\alpha}{2^{m-2}(m-2)!}\sum_{\ell=0}^{N}\widehat{u}_\ell(t)\int_{\mathbb{R}}w_\alpha(x,t)H_\ell(\alpha x)\,H_{m-2}(\alpha x)\,dx$$

$$= \frac{\alpha^2}{\sqrt{\pi}}\frac{1}{2^{m-2}(m-2)!}\sum_{\ell=0}^{N}\widehat{u}_\ell(t)\int_{\mathbb{R}}H_\ell(\zeta)H_{m-2}(\zeta)e^{-\zeta^2}\,d\zeta$$

$$= \frac{\alpha^2}{2^{m-2}(m-2)!\sqrt{\pi}}\sum_{\ell=0}^{N}\widehat{u}_\ell(t)\,2^{m-2}(m-2)!\sqrt{\pi}\delta_{\ell,m-2} = \alpha^2\widehat{u}_{m-2}(t).$$

Regarding the last integral, we split it into two parts:

$$\int_{\mathbb{R}}u_N\,\frac{\partial\phi_m}{\partial t}\,dx = \frac{1}{\sqrt{\pi}}\,\frac{1}{2^m\,m!}\sum_{\ell=0}^{N}\widehat{u}_\ell(t)\int_{\mathbb{R}}w_\alpha(x,t)H_\ell(\alpha x)\,\partial_t\Big(\alpha H_m(\alpha x)\Big)\,dx$$

$$= \frac{1}{\sqrt{\pi}}\,\frac{1}{2^m\,m!}\sum_{\ell=0}^{N}\widehat{u}_\ell(t)\int_{\mathbb{R}}w_\alpha(x,t)H_\ell(\alpha x)\left(\alpha' H_m(\alpha x)+\alpha\alpha' xH'_m(\alpha x)\right)dx$$

$$= \frac{1}{\sqrt{\pi}}\,\frac{1}{2^m\,m!}\,\frac{\alpha'}{\alpha}\sum_{\ell=0}^{N}\widehat{u}_\ell(t)\Big[\int_{\mathbb{R}}H_\ell(\zeta)H_m(\zeta)e^{-\zeta^2}\,d\zeta+\int_{\mathbb{R}}H_\ell(\zeta)\,\zeta H'_m(\zeta)e^{-\zeta^2}\,d\zeta\Big]$$

$$= \textbf{(I)}+\textbf{(II)}. \tag{62}$$

These are finally evaluated as follows:

$$\textbf{(I)} = \frac{1}{\sqrt{\pi}}\,\frac{1}{2^m\,m!}\,\frac{\alpha'}{\alpha}\sum_{\ell=0}^{N}\widehat{u}_\ell(t)\big(2^m\,m!\sqrt{\pi}\big)\delta_{\ell,m} = \frac{\alpha'(t)}{\alpha}\widehat{u}_m.$$

$$\begin{aligned}
\textbf{(II)} &= \frac{1}{\sqrt{\pi}}\,\frac{1}{2^m\,m!}\,\frac{\alpha'}{\alpha}\sum_{\ell=0}^{N}\widehat{u}_\ell(t)\left[\int_{\mathbb{R}} H_\ell(\zeta)\,\zeta H'_m(\zeta)e^{-\zeta^2}\,d\zeta\right]\\
&= \frac{1}{\sqrt{\pi}}\,\frac{1}{2^m\,m!}\,\frac{\alpha'}{\alpha}\sum_{\ell=0}^{N}\widehat{u}_\ell(t)\big(2^\ell\,\ell!\sqrt{\pi}\big)\left[2m(m-1)\delta_{m-2,\ell} + m\delta_{\ell,m}\right]\\
&= \frac{1}{\sqrt{\pi}}\,\frac{1}{2^m\,m!}\,\frac{\alpha'}{\alpha}\big(2^{m-2}\,(m-2)!\sqrt{\pi}\big)\left[2m(m-1)\widehat{u}_{m-2}(t) + \big(2^m\,m!\sqrt{\pi}\big)\,m\widehat{u}_m(t)\right]\\
&= \frac{\alpha'}{\alpha}\left[\frac{1}{2}\widehat{u}_{m-2}(t) + m\widehat{u}_m(t)\right]. \qquad (63)
\end{aligned}$$

The final result of all these computations is again the scheme (14).