Computer Physics Communications 280 (2022) 108474

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

Time-dependent Dirac Equation with Physics-Informed Neural Networks: Computation and Properties ☆

Emmanuel Lorin [a,b,*], Xu Yang [c]

[a] School of Mathematics and Statistics, Carleton University, Ottawa, K1S 5B6, Canada
[b] Centre de Recherches Mathématiques, Université de Montréal, Montréal, H3T 1J4, Canada
[c] Department of Mathematics, University of California, Santa Barbara, CA 93106, USA

A R T I C L E   I N F O

Article history:
Received 7 April 2022
Received in revised form 20 July 2022
Accepted 23 July 2022
Available online 1 August 2022

Keywords:
Physics-informed neural network
Dirac equation
Quantum relativistic physics
Scientific machine learning

A B S T R A C T

In this paper, we are interested in the computation of the time-dependent Dirac equation using physics-informed neural networks (PINNs), a new powerful tool in scientific machine learning avoiding the use of approximate derivatives of differential operators. PINNs search solutions in the form of parameterized (deep) neural networks, whose derivatives (in time and space) are performed by automatic differentiation. The computational cost comes from the need to solve high-dimensional optimization problems using stochastic gradient methods in the training the network with a large number of points analogues of the discretization points for standard PDE solvers. Specifically, we derive a PINNs-based algorithm and present some key fundamental properties when applied to the Dirac equations in different physical frameworks.

1. Introduction

In this paper, we are interested in computing the relativistic two-dimensional time-dependent Dirac equation (TDDE) using physics-informed neural network (PINN) algorithms [1–4]. PINN is a new machine-learning method for solving partial differential equations (PDEs) using neural networks (NNs), which takes its origin in the celebrated work of Lagaris [5]. Generally speaking, the strength of the PINN approach is its impressive flexibility, particularly regarding the dimensionality of the PDE under consideration, and its mathematical structure, at least for linear PDEs. However, solutions to nonlinear PDE systems may require much higher computational resources. We will see, for instance, that when solving the Dirac equation in flat or curved spaces, the structure of the PINN algorithm is identical, unlike standard PDE solvers, which require more technical attention in curved space (regarding the numerical stability, or the order of convergence). More specifically, a loss function is constructed as the norm of the PDE operator under consideration applied to the searched neural network. This brings the following advantages: i) PINN does not require the discretization of any differential operator; ii) is embarrassingly parallel; and iii) is easy to implement (at least using neural network libraries). As in any neural-network-based /machine learning algorithm, the training allows for the construction of a space-time function (neural network) by a minimization process (providing the weight and bias parameters) and can be evaluated/tested at any time and space. In particular, the computed network can be used as an initial space-time network for other IBVP involving TDDE. The derivatives of neural networks are performed exactly using automatic differentiation. The PINN algorithm applied to the space-time differential operator is not based on an evolution procedure but on the minimization of nonlinear functions thanks to stochastic gradient methods [6,7]. The counterpart is that the minimization algorithm usually does not converge to a global minimum. Hence, we have to make sure that the computed solution is close enough to the exact solution by choosing the optimized algorithm properly. Additional constraints (such as mass conservation) may be used to guide the minimization algorithm. Unlike, for instance, recent by Grobe et al. [8], the PINN algorithm presented below (at least by default) has no requirement on the structure of the solution to the Dirac equation. Here, we propose a pedagogical presentation of the algorithms, particularly without prior scientific machine learning knowledge, starting with introducing the TDDE under consideration. The intent of this paper is then to give some relevant information (pros and cons) on the use of PINN algorithms specifically for solving the TDDE.

☆ The review of this paper was arranged by Prof. N.S. Scott.
* Corresponding author.
E-mail addresses: elorin@math.carleton.ca (E. Lorin), xuyang@math.ucsb.edu (X. Yang).

https://doi.org/10.1016/j.cpc.2022.108474
0010-4655/© 2022 Elsevier B.V. All rights reserved.

Let us also mention that these past five years, an important effort has been put into the development of NN-based methods in quantum chemistry, more specifically for constructing the ground state of many-body Schrödinger Hamiltonians using variational quantum Monte-Carlo and/or diffusion Monte-Carlo; see for instance [9,10]. An analogue to the latter approaches would consist in implementing the imaginary time (or continuous gradient flow) method [11] to compute the ground state of the Schrödinger Hamiltonian using a PINN algorithm. Naturally, a PINN-Dirac solver in imaginary time would not be sufficient to directly compute the ground state of the Dirac Hamiltonian due to its unboundedness. An operator balance [12] can, for instance, be used to fix this issue. In this paper, we do not intend to study the general efficiency or accuracy of PINN algorithms, but rather to point out the benefits of using such an approach compared to standard Dirac equation solvers. We will see that automatic differentiation in space and time allows for avoiding stability, numerical dispersion (Fermion doubling), and numerical diffusion issues. It also allows for easily considering the Dirac Hamiltonian in "less standard" configurations, such as strained graphene where the coefficients are non-constant, without complexifying the overall algorithm.

1.1. *Time-dependent Dirac equation*

We consider the following two-dimensional time-dependent Dirac equation on $\Omega \times [0,T]$ [13] ($\Omega \subseteq \mathbb{R}^2$)

$$\mathrm{i}\partial_t \psi = -\big\{\sigma_x(\mathrm{i}c\partial_x + eA_x(\boldsymbol{x},t)) + \sigma_y(\mathrm{i}c\partial_y + eA_y(\boldsymbol{x},t)) - mc^2\sigma_z - V(\boldsymbol{x})I_2 - A_0(\boldsymbol{x},t)I_2\big\}\psi\,,$$

and initial condition $\psi(\boldsymbol{x},0) = \phi_0(\boldsymbol{x}) \in L^2(\Omega;\mathbb{C}^2)$, where c denotes the speed of light, m is the mass of the electron, and where Pauli's matrices are defined by

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\,,\ \sigma_y = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix} \text{ and } \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{1}$$

For Ω bounded, we impose (except if specified otherwise) null Dirichlet boundary condition on Ω. Function V represents an interaction potential, and (A_x, A_y), A_0 denotes an external space-time dependent electromagnetic field which is assumed to be given. In the following, we will also denote $\widetilde{\sigma}_y := -\mathrm{i}\sigma_y$. In the framework of this paper, the Dirac equation typically models relativistic fermions [13,14] possibly interacting with an external electromagnetic field. There exists an important literature on computational methods for solving this problem; see for instance [15–20].

Specifically, we are interested in the proof of feasibility, and relevance of the PINN method in the framework of relativistic quantum physics. In particular, the focus will be put on the behavior of this methodology regarding some fundamental properties from the computational and physical points of view: scaling when c increasing, numerical dispersion (Fermion doubling [21]), and preservation of the L^2-norm.

Using the PINN algorithm for the TDDE allows to avoid the space&time derivative approximation with a loss function which has a *natural* structure. In particular, we can avoid the usual stability constraints on the time step, as it is no more necessary to discretize in time [11,22–24,15]. The presence of the external electromagnetic field would for instance complexify the stable and accurate discretization of the Dirac equation with standard approximate PDE solvers [15,24]. Hereafter, for computational convenience, we rewrite the Dirac equation in the form of a *real* 4-equation system

$$\mathcal{H}_D(\boldsymbol{x},t)\Phi \quad := \quad \partial_t\Phi + H_x\partial_x\Phi + H_y\partial_y\Phi + C(\boldsymbol{x},t)\Phi\,, \tag{2}$$

where $\Phi = (\psi_R, \psi_I) \in L^2(\Omega\times[0,T];\mathbb{R}^4)$ the real and imaginary parts of the Dirac equation solution $\psi \in L^2(\Omega\times[0,T];\mathbb{C}^2)$, and where

$$H_x = c\begin{pmatrix} \sigma_x & 0_2 \\ 0_2 & \sigma_x \end{pmatrix}, \quad H_y = c\begin{pmatrix} 0_2 & -\widetilde{\sigma}_y \\ \widetilde{\sigma}_y & 0_2 \end{pmatrix},$$

and where $C(\boldsymbol{x},t)$ is given by

$$\begin{pmatrix} 0_2 & -V(\boldsymbol{x})I_2 - mc^2\sigma_z + eA_x(\boldsymbol{x},t)\sigma_x + eA_y(\boldsymbol{x},t)\widetilde{\sigma}_y \\ V(\boldsymbol{x})I_2 + mc^2\sigma_z - eA_x(\boldsymbol{x},t)\sigma_x - eA_y(\boldsymbol{x},t)\widetilde{\sigma}_y & 0_2 \end{pmatrix}.$$

We denote by $\Phi_0 = (\phi_0, \mathbf{0})^T \in L^2(\Omega;\mathbb{R}^4)$ the initial condition. Let us recall that the Dirac equation is a Hermitian linear first-order hyperbolic system [25]. In the following, we will also denote $H_D(\boldsymbol{x},t) := H_x\partial_x + H_y\partial_y + C(\boldsymbol{x},t)$.

1.2. *Neural networks*

We here recall the basics of neural network-based algorithms for solving PDE. The principle consists in searching for the solution to the PDE under consideration, in the form of a parameterized deep neural network and to search for the corresponding parameters by minimizing a loss function. Denoting the neural network $N(\boldsymbol{\theta},\boldsymbol{x})$ with $\boldsymbol{x} = (x_1,\cdots,x_d) \in \Omega \subseteq \mathbb{R}^d$ and by $\boldsymbol{\theta}$ the unknown parameters, neural networks usually read (for 1 hidden layer, machine learning)

$$N(\boldsymbol{\theta},\boldsymbol{x}) \quad = \quad \textstyle\sum_{i=1}^{H} v_i\sigma_i\big(\sum_{j=1}^{d}\theta_{ij}x_j + u_i\big)\,, \tag{3}$$

where $\{\sigma_i\}_{1\leqslant i\leqslant H}$ are the sigmoid transfer functions, and H is the number of sigmoid units (Neurons), $\{\theta_{ij}\}_{ij}$ are the weights and $\{u_i\}_i$ the bias. When considering several hidden layers (deep learning), we have then to compose functions of the form (3), [26]. That is,

$$N = \mathcal{N}_p \circ \mathcal{N}_{p-1} \circ \cdots \mathcal{N}_1 \circ \mathcal{N}_0\,,$$

where for $0 \leqslant r \leqslant p$, $\mathcal{N}_r$ is defined from $\mathbb{R}^{a_r}$ (with $a_r \in \mathbb{N}$) to $\mathbb{R}^{a_{r+1}}$ by $\sigma_r(\Theta_r X_r + b_r)$, σ_r is an activation function, $X_r \in \mathbb{R}^{a_r}$ and where $(a_0,\cdots,a_{p+1})$ where $p+1$ layers are considered. The layer $r=0$ is the input layer and $r=p+1$ is the output layer, such that $a_0 = a_{p+1} = m$. We refer to [27] details about neural networks.

In this paper, we propose to adapt the PINN algorithm for the Dirac equation in different frameworks, such as relativistic quantum physics, and strained graphene. Beyond the simple derivation of the algorithm and its computational complexity, we are also interested in some fundamental properties of this solver. In particular, some crucial physical constraints such as the L^2-norm conservation of the wavefunction or the fermion-doubling issue [24].

1.3. Organization

This paper is organized as follows. In Section 2, we introduce the PINN algorithm in the framework of the TDDE, and some computational and mathematical properties of the method. Section 3 is dedicated to numerical experiments for the PINN-Dirac algorithm. Some concluding remarks are finally proposed in Section 4.

2. PINN algorithm for the Dirac equation

In this section, we present the principle of the PINN algorithm applied to the TDDE. We search for an approximate solution to this evolution PDE in the form of a space-time dependent vector valued deep neural network $(N_1, N_2, N_3, N_4)^T = \boldsymbol{N} \in L^2(\Theta \times \Omega \times [0,T]; \mathbb{R}^4)$, where $\Theta \subset \mathbb{R}^p$ denotes the search space. In order to include the initial condition, we can search for NN in the form

$$\boldsymbol{N} \longleftarrow \boldsymbol{\Phi}_0 + f(t)\boldsymbol{N}\, , \text{with} \quad f(0) = 0.$$

For example, one may take $f(t) = t$ as in [5]. Alternatively, it is possible to pick other functions allowing to consider larger times such as $f(t) = (1 - e^{-\gamma t})/\gamma$ for some free parameters $\gamma > 0$. In particular, when t goes to infinity, the contribution of the initial data in the loss function is no more negligible; this allows to avoid the convergence of the PINN algorithm to a null solution. For long physical times, it may be necessary to impose a L^2-norm constraint: $\|\boldsymbol{N}(\boldsymbol{\theta}, \cdot, t)\|_{L^2(\Omega)} = \|\boldsymbol{N}_0(\boldsymbol{\theta}, \cdot)\|_{L^2(\Omega)} = 1$. In the latter case, the PINN algorithm (see [1–4]) hence consists in computing for some free parameter $\lambda \geqslant 0$

$$\overline{\boldsymbol{\theta}} = \text{argmin}_{\boldsymbol{\theta}\in\Theta}\big\{\big\|\mathcal{H}_D(\cdot,\cdot)\boldsymbol{N}(\boldsymbol{\theta},\cdot,\cdot)\big\|^2_{L^2(\Omega\times(0,T))} + \lambda\big\||\boldsymbol{N}(\boldsymbol{\theta},\cdot,\cdot)|_{\ell^2} - 1\big\|^2_{L^2(\Omega\times(0,T))}\big\}\, ,$$

where $\mathcal{H}_D$ is defined in (2). Additional constraints can naturally easily be added to force some conditions. A fundamental element of the algorithm is the automatic differentiation of the neural network which allows to perform exactly the derivative of NN. The counterpart is the regularity of the overall solution is naturally imposed by the regularity of the neural network. With such minimization, we ensure that the searched function is indeed an approximate solution to the Dirac equation, and that it approximately conserves the L^2-norm. An alternative approach based on time-domain decomposition is also proposed in Section 3. In this paper, we will focus on the computation of the density

$$\rho_N(\boldsymbol{x}, t) = \sum_{i=1}^{4} |N_i(\overline{\boldsymbol{\theta}}, \boldsymbol{x}, t)|^2\, . \tag{4}$$

It is also interesting to notice that in the above algorithm, the searched neural network is a space- and time-dependent function. In other words, the initial condition is mathematically considered as a Dirichlet boundary condition in space-time domain. As a consequence, we get an approximate solution of the TDDE at any time and any point in space. The inclusion of more complex boundary conditions will require to add some constraints to the loss function.
The PINN algorithm requires the computation of the NN parameters obtained thanks to the minimization of the loss function (*training process*):

- We first construct (usually refers as *forward pass*) the loss functions at some *input data*

  $$\{(\boldsymbol{x}_j, t_n)\}_{(j,n)\in\mathbb{N}^{(i)}_{\boldsymbol{x}}\times\mathbb{N}_t} \quad (\text{resp. } \{(\boldsymbol{x}_j, t_n)\}_{(j,n)\in\mathbb{N}^{(e)}_{\boldsymbol{x}}\times\mathbb{N}_t})$$

  in the interior (resp. exterior) of the space-time domain. We denote by $\mathcal{N}^{(i)}_{\boldsymbol{x}}$ (resp. $\mathcal{N}^{(e)}_{\boldsymbol{x}}$ and $\mathcal{N}_t$) the number of elements in the set denoted by $\mathbb{N}^{(i)}_{\boldsymbol{x}}$ (resp. $\mathbb{N}^{(e)}_{\boldsymbol{x}}$ and $\mathbb{N}_t$). For instance, with null Dirichlet boundary conditions the loss function (without the normalization constraint) is numerically minimized in the form

  $$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) \quad = \quad & \tfrac{1}{\mathcal{N}_{\boldsymbol{x}}\mathcal{N}_t}\textstyle\sum_{(j,n)\in\mathbb{N}^{(i)}_{\boldsymbol{x}}\times\mathbb{N}_t}\big|\mathcal{H}_D(\boldsymbol{x}_j, t_n)\boldsymbol{N}(\boldsymbol{\theta}, \boldsymbol{x}_j, t_n)\big|^2_2 \\ & + \tfrac{1}{\mathcal{N}^{(e)}_{\boldsymbol{x}}\mathcal{N}_t}\textstyle\sum_{(j,n)\in\mathbb{N}^{(e)}_{\boldsymbol{x}}\times\mathbb{N}_t}\big|\boldsymbol{N}(\boldsymbol{\theta}, \boldsymbol{x}_j, t_n)\big|^2_2\, .\end{aligned}$$

  This process can be very computationally complex, although its parallelization is straightforward.
- Minimization (*backward pass*) of the loss function is performed using stochastic gradient descent (SGD) method [28,6,7]. The interest of the SGD is to deal with (several) lower dimensional optimization problems allowing to avoid the use of a deterministic gradient method on a very high dimensional search space. The latter requires to fix some parameters such as the epochs (corresponding to the number of iterations in the model training) and a learning rate.
- Standard PDE solvers require the use of a time step less than $1/mc^2$ in order to precisely approximate the mass-term [15] and the phenomenon called *Zitterbewegung* [13]. Practically, this often limits numerical simulations for TDDE to very short physical times. PINN algorithms can not circumvent this issue as the TDDE solution, even if it is not constructed on an evolution process. More specifically, at a final time T, the number of training data in the t-direction should scale in Tmc^2. Hence, there is a need for dealing with a very high dimensional minimization problem when c and T are large. Assuming that the solution does not contain high wavenumbers (space frequency), and in order to avoid an overfitting in space, however we can take $\mathbb{N}^{(i,e)}_{\boldsymbol{x}}$ small. Similarly, in order to avoid underfitting in time, we need to take $\mathbb{N}^{(i,e)}_t$ large enough.

- Practically, we compute the solution to the Dirac equation on a bounded spatial subset $\Omega = (-L, L)^2$ of $\mathbb{R}^2$, and we impose Dirichlet boundary conditions at $(\pm L, y)$ and $(x, \pm L)$ which can easily be imposed through additional constraints on the loss function.
- The training of the network will allow for a parameterization of the space-time solution of the TDDE. This parameterized function makes it possible to evaluate the approximate solution at any space-time location.
- A trained network, for a given IVP involving a Dirac equation, can also be used as an initial guess for approximating the solution to other IVP with the same initial data but perturbed Dirac operator, such as initializing a network for solving the Dirac equation in curved space and using the space-time solution to the Dirac equation in flat space as an initial guess.

In order to improve the accuracy or precision of the NN-based solver, it is naturally possible to impose some additional constraints (e.g. symmetry) on the structure of the functions involved in the neural networks rather than imposing those constraints through penalization terms, which would complexify the optimization.

Remark 2.1. Extension to TDDE in curved space. This remark is devoted to the application of the PINN to the TDDE in curved space. The latter is in particular used to model strained graphene surfaces [29–33]. In the case of stationary surfaces, the TDDE has space-dependent Pauli/Dirac matrices. The corresponding Dirac equation models non-relativistic electrons in the vicinity of Dirac points (low energy limit). Specifically for a general smooth two-dimensional surface, and neglecting the non-diagonal terms of the metric tensor associated to the surface [31], the systems reads

$$\mathcal{H}_D(\boldsymbol{x})\psi(\boldsymbol{x},t) = 0\,,$$

where

$$\mathcal{H}_D(\boldsymbol{x}) = \mathrm{i}\partial_t + \mathrm{i}\frac{v_F}{\sqrt{\rho(\boldsymbol{x})}}\Big\{\sigma_x\left[\partial_x + \Omega_x(\boldsymbol{x}) + \mathrm{i}A_x(\boldsymbol{x})\right] + \sigma_y\left[\partial_y + \Omega_y(\boldsymbol{x}) + \mathrm{i}A_y(\boldsymbol{x})\right]\Big\}\,,$$

with affine spin connection

$$\Omega_x(\boldsymbol{x}) = \frac{\mathrm{i}}{4}\frac{\rho_y(\boldsymbol{x})}{|\rho(\boldsymbol{x})|}\sigma_z, \;\; \Omega_y(\boldsymbol{x}) = -\frac{\mathrm{i}}{4}\frac{\rho_x(\boldsymbol{x})}{|\rho(\boldsymbol{x})|}\sigma_z\,,$$

where ρ is a smooth surface-dependent function, v_F is the Fermi velocity for unstrained graphene surface, and $\boldsymbol{A} = (A_x, A_y)$ is a (pseudo)magnetic field. We refer to [31] for details. Interestingly, the PINN again simply reads

$$\overline{\boldsymbol{\theta}} = \mathrm{argmin}_{\boldsymbol{\theta}\in\Theta}\big\|\mathcal{H}_D(\cdot)\boldsymbol{N}(\boldsymbol{\theta},\cdot,\cdot)\big\|^2_{L^2(\Omega\times(0,T))}\,.$$

3. Properties and numerical experiments

In this section, we discuss several important properties of the PINN-Dirac algorithm along with illustrating numerical experiments. The latter will be solved on a bounded domain with null Dirichlet boundary conditions. The implementation was performed using `DeepXDE` [34] and `tensorflow` [35] which are very flexible and easy-to-use libraries. In the tests below, we have taken $\lambda = 0$, $c = 1$ and we use tanh as transfer function. The `python` codes will be made available on `github`.
We are first interested in this subsection in some important numerical and physical properties of the PINN solver for the Dirac equation, in particular in the framework of relativistic quantum physics. More specifically, we discuss several important questions.

3.1. Numerical dispersion (Fermion doubling)

Numerical dispersion refers in the quantum relativistic and high energy physics literatures, and more specifically in gauge field theory as *Fermion doubling* [24,36,21]. The latter occurs when the dispersion relation is not exactly satisfied, which generates artificial modes, and leads to chiral symmetry breaking [24]. Standard real space methods usually generate these artificial fermionic states, while only one state exists at the continuous level; see [24] and associated references. Mathematically those structures appear in the graph of the eigenvalues of the amplification factor of the scheme (in Fourier-space). While the amplification factor of the continuous systems possesses 2 eigenvalues with conical graphes $\pm\sqrt{c^2|\mathbf{k}|^2 + m^2c^4}$, real approximation methods have amplification factors (see [37]) usually having eigenvalues with sinusoidal contribution, hence generating a periodic conical structures (in Fourier space). We refer the interested readers to [21], where this question is particularly well addressed for real space approximate solvers. In order to discuss this question with the PINN-framework, we consider $\psi(\boldsymbol{x},t) \in \mathbb{C}^2$ solution to

$$\mathrm{i}\partial_t\psi - \mathrm{i}c\boldsymbol{\sigma}\cdot\nabla\psi + mc^2\sigma_z\psi = \boldsymbol{0}\,,$$

where we have denoted $\boldsymbol{\sigma} = (\sigma_x, \sigma_z)$. The Fourier transform in space is denoted $\widehat{\psi}(\boldsymbol{k},t)$ with $\boldsymbol{k} = (k_x, k_y)^T$, and $\widehat{\psi}_0(\boldsymbol{k})$ denotes the Fourier transform of the initial data. Formally, we get

$$\widehat{\psi}(\boldsymbol{k},t) = G(\boldsymbol{k},t)\widehat{\psi}_0(\boldsymbol{k})\,,$$

where the 2×2 (amplification) matrix reads

$$G(\boldsymbol{k},t) = \exp\Big[-\mathrm{i}\big(ct\boldsymbol{\sigma}\cdot\boldsymbol{k} + mc^2\sigma_z\big)\Big]\,.$$

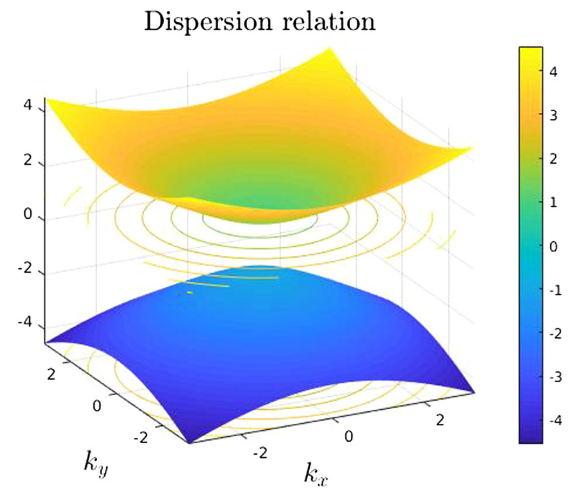


Fig. 1. Exact dispersion relation: graph of $\mu_{1,2}$ in (5). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

The corresponding eigenvalues $h_{\pm}(\boldsymbol{k}) = \exp\big(\mp \mathrm{i}t\sqrt{|\boldsymbol{k}|^2c^2+m^2c^4}\big)$ with absolute value equal 1, corresponding to the conservation of mass (or L^2-norm). Rewriting $G = PDP^{-1}$, where $D = \mathrm{diag}(h_-, h_+)$ and the transition matrix reads

$$P(\boldsymbol{k}) = \begin{pmatrix} -\sqrt{|\boldsymbol{k}|^2c^2+m^2c^4}+mc^2 & ck_x - \mathrm{i}ck_y \\ ck_x+\mathrm{i}ck_y & \sqrt{|\boldsymbol{k}|^2c^2+m^2c^4}-mc^2 \end{pmatrix}.$$

We set $\widetilde{\psi}(\boldsymbol{k},t) := P^{-1}(\boldsymbol{k})\widehat{\psi}(\boldsymbol{k},t)$ so that $\widetilde{\psi} = D\widetilde{\psi}_0$, and

$$\begin{array}{rcl} \widetilde{\psi}_1(\boldsymbol{k},t) & = & \exp\big(+\mathrm{i}t\sqrt{|\boldsymbol{k}|^2c^2+m^2c^4}\big)\widetilde{\psi}_{0;1}(\boldsymbol{k}), \\ \widetilde{\psi}_2(\boldsymbol{k},t) & = & \exp\big(-\mathrm{i}t\sqrt{|\boldsymbol{k}|^2c^2+m^2c^4}\big)\widetilde{\psi}_{0;2}(\boldsymbol{k}). \end{array}$$

We then define μ_i $(i = 1,2)$

$$\mu_i(\boldsymbol{k},\boldsymbol{t}) = \mathrm{i}\log\Big[\frac{\widetilde{\psi}_i(\boldsymbol{k},t)}{\widetilde{\psi}_{0;i}(\boldsymbol{k},t)}\Big] = (-1)^i t\sqrt{|\boldsymbol{k}|^2c^2+m^2c^4}. \quad (5)$$

The graphs of $\mu_{1,2}$ are reported in Fig. 1.

The objective is hence to compare the above dispersion relation with the one obtained numerically thanks to the PINN algorithm. In this goal, we mimic the above computation in the PINN framework for a NN solution $\boldsymbol{N}(\overline{\boldsymbol{\theta}},\boldsymbol{x},t) \in \mathbb{R}^4$. In order to properly reproduce the above calculations, we set $\boldsymbol{M} = (M_1, M_2) := (N_1 + \mathrm{i}N_3, N_2 + \mathrm{i}N_4)$ which takes it values in $\mathbb{C}^2$ and which is a PINN approximation to ψ, as described in Section 2. We denote by $\widehat{\boldsymbol{M}}$ the (discrete) Fourier transform in space of $\boldsymbol{M}$, and we set $\widetilde{\boldsymbol{M}}(\overline{\boldsymbol{\theta}},\boldsymbol{k},t) = P^{-1}(\boldsymbol{k})\boldsymbol{M}(\overline{\boldsymbol{\theta}},\boldsymbol{k},t)$ so that $\widetilde{\boldsymbol{M}} = D\widetilde{\boldsymbol{M}}_0$. We then propose to compare μ_i with ζ_i for $i = 1,2$, defined as follows

$$\zeta_i(\boldsymbol{k},t) = \mathrm{i}\log\Big[\frac{\widetilde{\boldsymbol{M}}_i(\boldsymbol{k},t)}{\widetilde{\boldsymbol{M}}_{0;i}(\boldsymbol{k})}\Big]. \quad (6)$$

More specifically for fixed t, we will represent the graph of $\boldsymbol{k} \mapsto \zeta_i(\boldsymbol{k},t)$.

Experiment 1. We propose the following experiment with $m = 1$, $c = 1$, $V = 0$ in (2). We report the graph of μ_i versus the one of ζ_i for $i = 1,2$. The space-time domain is $(-8,8) \times [0,2]$ and we impose Dirichlet boundary condition. Initially we take

$$\phi_0(\boldsymbol{x}) = \exp\big(-2\|\boldsymbol{x}\|_2^2 + \mathrm{i}\boldsymbol{k}_0\cdot\boldsymbol{x}\big)(1,0)^T, \quad (7)$$

where $\boldsymbol{k}_0 = (1,1)^T$. We takes 1.5×10^4 epochs (training iterations) and 5×10^4 input data. The chosen network possesses 8 layers and 50 neurons. The converged solution is reached with a train loss of 9×10^{-7} and test loss of 1.18×10^{-6}. We report in Fig. 2 the graph of ζ_1 (Left) and ζ_2 (Right) at $T = 2$ obtained as described above.

This example shows that the PINN-method is free from fermion doubling; the latter would indeed correspond to the generation of artificial modes (typically in the form of periodic conical structures regularly spaced; see [21]).

3.2. L^2-norm stability and conservation of the density for long physical times

The preservation of the L^2-norm of the solution is a fundamental physical property, which can be interpreted as a mass conservation; that is, for any $t \geqslant 0$, we expect that the approximate solution satisfies $\|\boldsymbol{N}(\overline{\boldsymbol{\theta}},\cdot,t)\|_{L^2((\Omega);\mathbb{R}^4)} = 1$. In order to preserve the L^2-norm, it is possible to directly impose a constraint within the loss function as described above. However, let us recall that in the above setting, the initial condition is imposed through a condition on the boundary $\Omega\times\{t=0\}$. Hence, without imposing this additional constraint, only the boundary/initial condition ensures that the PINN solution to the Dirac equation will not be null. As a consequence, for long physical

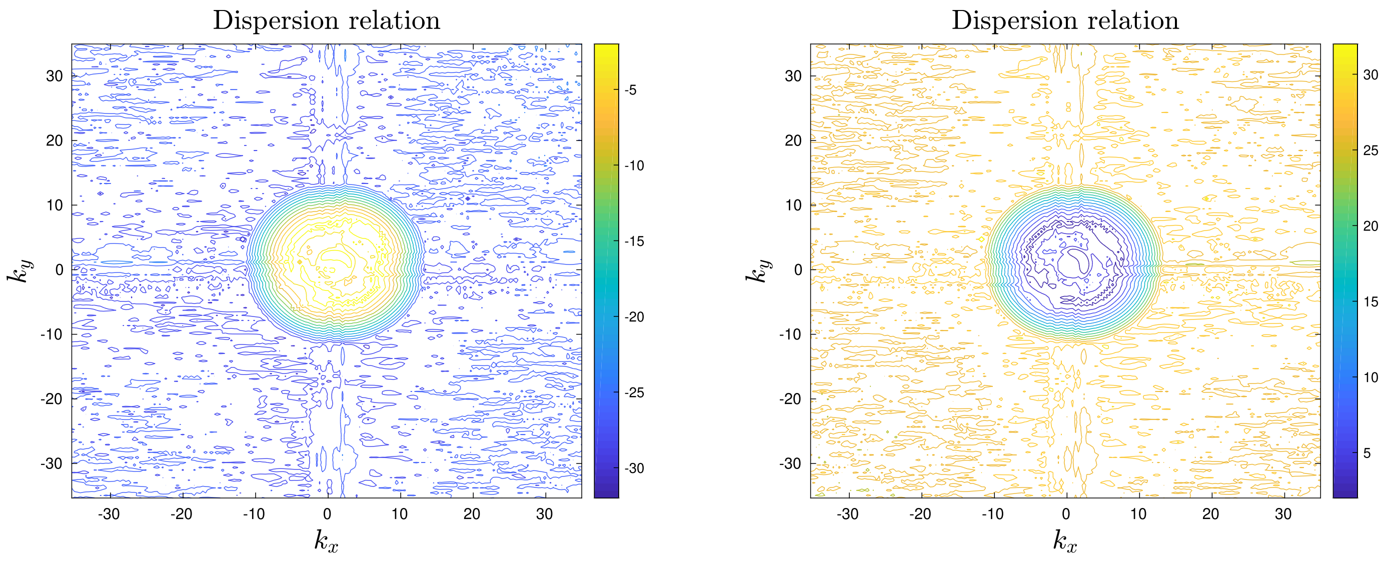


Fig. 2. Experiment 1. Numerical dispersion relation (6): graphs of ζ_1 (Left) and ζ_2 (Right).

times (hence requiring a very large number of input data, in particular in the time direction), the minimization of the train loss function may lead to a solution with L^2-norm close to zero. This was already observed in [38] where it was the first time implemented for a one-dimensional PINN-Dirac algorithm.

Rather than or in addition to the constraint on the loss function forcing the L^2-norm of the solution to 1, we propose a natural alternative approach, for long physical times. This simple idea consists in decomposing the time domain as follows $[0,T]=\cup_{i=0}^{N}[t_i,t_{i+1}]$ with $t_0=0$ and $t_{N+1}=T$ and with $\Delta t = t_{i+1}-t_i$ small enough to unsure the L^2-norm preservation during each "small" space-time subdomain $\Lambda_i := \Omega\times[t_i,t_{i+1}]$. On each of these space-time Λ_i subdomains, we apply the same strategy/method as above. However in this case, most of the training is performed in the spatial directions. Assuming Φ_i given (and computed on Λ_{i-1}), we solve on Λ_i

$$\begin{array}{rcl} \mathcal{H}_D(\boldsymbol{x},t)\Phi_{i+1} & = & 0 \\ \Phi_{i+1}(\cdot,t_i) & = & \Phi_i(\cdot,t_i)\,, \end{array} \tag{8}$$

by minimizing on each interval (t_i,t_{i+1}) with initial data $\boldsymbol{N}(\overline{\boldsymbol{\theta}}_i,\cdot,t_i)$

$$\overline{\boldsymbol{\theta}}_{i+1} \;=\; \mathrm{argmin}_{\boldsymbol{\theta}\in\Theta}\Big\{\big\|\mathcal{H}_D(\cdot,\cdot)\boldsymbol{N}(\boldsymbol{\theta},\cdot,\cdot)\big\|^2_{L^2(\Lambda_i)} + \lambda\big\||\boldsymbol{N}(\boldsymbol{\theta},\cdot,\cdot)|_{\ell^2}-1\big\|^2_{L^2(\Lambda_i)}\Big\}\,, \tag{9}$$

where $\boldsymbol{N}(\overline{\boldsymbol{\theta}}_{i+1},\cdot,\cdot)$ is a neural network approximating the solution Φ_{i+1}. Theoretically, this decomposition in time does not change the overall solution if it were performed exactly by the PINN algorithm. That is $\Phi_{N+1}=\Phi$ where Φ is the exact solution to (2). Hence, we just need to make sure that, at least on each interval the L^2-norm of Φ_i is preserved. Practically, we can also enforce the normalization of the wavefunction at t_{i+1}.

Notice that the proposed time-decomposition also allows for introducing some additional *learning* in the computation process. Indeed, denoting by $\mathcal{L}$ the loss function to minimize, gradient descent iterations at time iteration say $i+1$, read

$$\boldsymbol{\theta}^{(\ell+1)}_{i+1} = \boldsymbol{\theta}^{(\ell)}_{i+1} - \nu\nabla\mathcal{L}(\boldsymbol{\theta}^{(\ell)}_{i+1})\,, \tag{10}$$

where ℓ is the gradient iteration index, ν is the learning rate and where $\boldsymbol{\theta}^{(0)}_{i+1}$ is given. Unlike standard PDE solvers whose complexity is typically independent of the time iteration, the NN-based algorithm benefits from the use of small time domains. Indeed, from $\boldsymbol{N}(\overline{\boldsymbol{\theta}}_i,\cdot,\cdot)$ the converged network at time t_i, we can initiate (10) with $\boldsymbol{\theta}^{(0)}_{i+1}=\overline{\boldsymbol{\theta}}_i$. We then expect a much faster convergence of the gradient descent method. More specifically, for Δt small enough, and as $\boldsymbol{N}$ is smooth, $\|\boldsymbol{N}(\overline{\boldsymbol{\theta}}_{i+1},\cdot,\cdot)-\boldsymbol{N}(\overline{\boldsymbol{\theta}}_i,\cdot,\cdot)\|$ will in principle, be small by continuity. Then

$$\mathcal{L}(\overline{\boldsymbol{\theta}}_{i+1}) \;=\; \mathcal{L}(\overline{\boldsymbol{\theta}}_i) + \nabla\mathcal{L}(\overline{\boldsymbol{\zeta}}_{i+1})^T(\overline{\boldsymbol{\theta}}_{i+1}-\overline{\boldsymbol{\theta}}_i)\,,$$

for some $\overline{\boldsymbol{\zeta}}_{i+1}$, so that

$$\big\|\overline{\boldsymbol{\theta}}_{i+1}-\overline{\boldsymbol{\theta}}_i\big\| \;=\; \big\|\nabla\mathcal{L}(\overline{\boldsymbol{\zeta}}_{i+1})\big\|^{-1}\big|\mathcal{L}(\overline{\boldsymbol{\theta}}_{i+1})-\mathcal{L}(\overline{\boldsymbol{\theta}}_i)\big|\,, \tag{11}$$

is expected to be small, the convergence of the gradient descent is fast.

Remark 3.1. Alternatively to the approach proposed above, it is also possible to apply the PINN algorithm in space only. In the latter case, we however need to approximate the time derivative in $\mathcal{H}_D$. In this goal, we introduce discrete times $t_0<t_1<\cdots<t_n<t_{n+1}<\cdots$. Assuming that $\boldsymbol{N}^n(\overline{\boldsymbol{\theta}}^n,\boldsymbol{x})$ is a known function (computed at previous time iteration), we write

$$\boldsymbol{N}^{n+1}(\boldsymbol{\theta},\boldsymbol{x}) = \boldsymbol{N}^n(\overline{\boldsymbol{\theta}}^n,\boldsymbol{x}) + \int_{t_n}^{t_{n+1}} H_D(\boldsymbol{x},s)\boldsymbol{N}(\boldsymbol{\theta},\boldsymbol{x},s)ds\,.$$

It is then necessary to approximate the time-integral. We propose to use a trapezoidal quadrature leading to

$$\Big(I - \frac{t_{n+1}-t_n}{2}H_D(\boldsymbol{x},t_{n+1})\Big)\boldsymbol{N}^{n+1}(\boldsymbol{\theta},\boldsymbol{x}) = \Big(I + \frac{t_{n+1}-t_n}{2}H_D(\boldsymbol{x},t_n)\Big)\boldsymbol{N}^n(\overline{\boldsymbol{\theta}}^n,\boldsymbol{x})\,.$$

Hence the PINN-in space algorithm consists in minimizing the following loss (training) function

$$\mathcal{L}^{n+1}(\boldsymbol{\theta}) = \Big\|\Big(I - \frac{t_{n+1}-t_n}{2}H_D(\cdot,t_{n+1})\Big)\boldsymbol{N}^{n+1}(\boldsymbol{\theta},\cdot) - \Big(I + \frac{t_{n+1}-t_n}{2}H_D(\cdot,t_n)\Big)\boldsymbol{N}^n(\overline{\boldsymbol{\theta}}^n,\cdot)\Big\|_{L^2(\Omega)}\,.$$

Interestingly, and unlike standard PDE implicit solvers, the solution is not updated at time t_{n+1} by directly solving a linear system, but by minimizing the loss function. The approximate solution at time t_{n+1} is given by:

$$\boldsymbol{N}^{n+1}\big(\text{argmin}_{\boldsymbol{\theta}\in\Theta}\mathcal{L}^{n+1}(\boldsymbol{\theta}),\boldsymbol{x}\big)\,.$$

Notice that if we assume that V and $\boldsymbol{A}, A_0$ are null, H_D is a constant operator and by Parseval's identity and hyperbolicity of the Dirac Hamiltonian, we have

$$\|\boldsymbol{N}^{n+1}(\overline{\boldsymbol{\theta}}^{n+1},\cdot)\|_{L^2(\mathbb{R}^4)} \quad\approx\quad \Big\|\Big(I - \frac{t_{n+1}-t_n}{2}H_D\Big)^{-1}\Big(I + \frac{t_{n+1}-t_n}{2}H_D\Big)\boldsymbol{N}^n(\overline{\boldsymbol{\theta}}^n,\cdot)\Big\|_{L^2(\mathbb{R}^4)}$$

The latter can potentially be relaxed the requirement on the conservation of the L^2-norm, if the initial NN-based density has a L^2-norm equal to 1. The main interest compared to using a standard PDE solver, is indeed the fact that we avoid the computation of a very high dimensional linear system if we search for a very accurate/precise solution (corresponding to a fine grid in space).

Experiment 2. In order to illustrate the principle of the time-domain decomposition approach, we need to check if, on a sufficiently small time interval the norm of the density, can be properly preserved in (2). Indeed, at the end of the computation, the solution at final time can then be used as initial condition for the IBVP on the next time interval. We propose the following illustration, with $t_1 - t_0 = 10^{-1}$. As the time interval is small enough etc, we can naturally select a relatively small number of input data. We represent in this experiment, the L^2-norm of the density as a function of time $\{(t, \sum_{i=1}^4 |N_i(\cdot,t)|^2_{L^2(\Omega);\mathbb{R}}),\ t\in(0,10^{-1})\}$. The computational domain in space is $\Omega = (-5,5)$, and we use null Dirichlet boundary conditions. We select a network with 4 hidden layers and 30 neurons, and the learning rate is taken equal to 10^{-3} and epoch size equal to 100. The initial data is a Gaussian (7). We report in Fig. 3 (Top) the L^2-norm of the density as a function of time, when we successively use 50, 100, 200 and 500 input data in total. This shows that, at least taking a large enough number of input data the PINN algorithm allows for a good conservation of the overall L^2-norm. Thanks to the strategy developed in (8), (9), it is possible to ensure the norm conservation on very long physical times by decomposing the overall time-domain in small time-subdomains.

Experiment 2bis. In the second example, we propose to better illustrate the above time-domain decomposition with the same data as above, but we consider 200 input data, 3 hidden layers and 20 neurons. The computation is performed on the time interval $[0,0.5]$ which is this time decomposed in 10 time-subdomains of length 5×10^{-2}. We report in Fig. 3 (Bottom) the L^2 of the overall approximate wavefunction as a function of time (with normalization at t_i^+). In order to better visualize the 10 subdomains, we use vertical dashed lines on the figures. We observe that thanks to this simple decomposition it is possible to maintain the L^2-norm close to one during the evolution.

3.3. Computational complexity

Let us now discuss the computational complexity of the PINN approach (in its simplest form) versus a standard implicit real space (finite volume, finite difference, finite element) PDE solver on a $N_{\boldsymbol{x}}$ spatial grid points with N_t time iterations. Hereafter, we do not present a fine estimate of the computational complexity based on the most advanced parallel algorithms either for real-space or NN-based algorithms. In both cases, it is obviously possible to largely improve the complexity. Instead the objective is to show the general scaling in the most simple setting. The overall computational complexity of an implicit real space solver is $O(N_tN_{\boldsymbol{x}}^{\ell})$, where $1<\ell\leqslant 3$ depends on the sparsity of linear systems involved in the spatial discretization. We denote by $\mathcal{N}_{\boldsymbol{x}}$ (resp. $\mathcal{N}_t$) the total number of interior and boundary spatial (resp. temporal) input data, and by $\mathcal{N}_{\boldsymbol{\theta}}$ the dimension of the search space; the latter depends on the depth of the NN (number of hidden layers and neurons). Naturally, as we search for space-time dependent NN, the dimension of the search space must be increased accordingly, compared to the space-only NN, as described in Remark 3.1. In the following, let us denote by k_G the total number of gradient iterations (epochs) involved in the training. Hereafter, $N_{\boldsymbol{x}}N_t$ will represent the total number of input data. Practically the forward and backward passes on mini-batches are combined to reach a local minimum of the loss function.

- The construction of the loss function (forward pass) is a linear process requiring a total of $O(k_G\mathcal{N}_{\boldsymbol{x}}\mathcal{N}_t\mathcal{N}_{\boldsymbol{\theta}})$ operations.
- The explicit gradient method update (backward pass) to minimize the loss function also requires a total of $O(k_G\mathcal{N}_{\boldsymbol{x}}\mathcal{N}_t\mathcal{N}_{\boldsymbol{\theta}})$ operations, [28,6,7]. Implicit gradient solvers would require $O(k_G\mathcal{N}_{\boldsymbol{x}}\mathcal{N}_t\mathcal{N}_{\boldsymbol{\theta}}^{\alpha})$ operations, for some $1\leqslant\alpha\leqslant3$ but with a larger learning rate than explicit gradient methods.

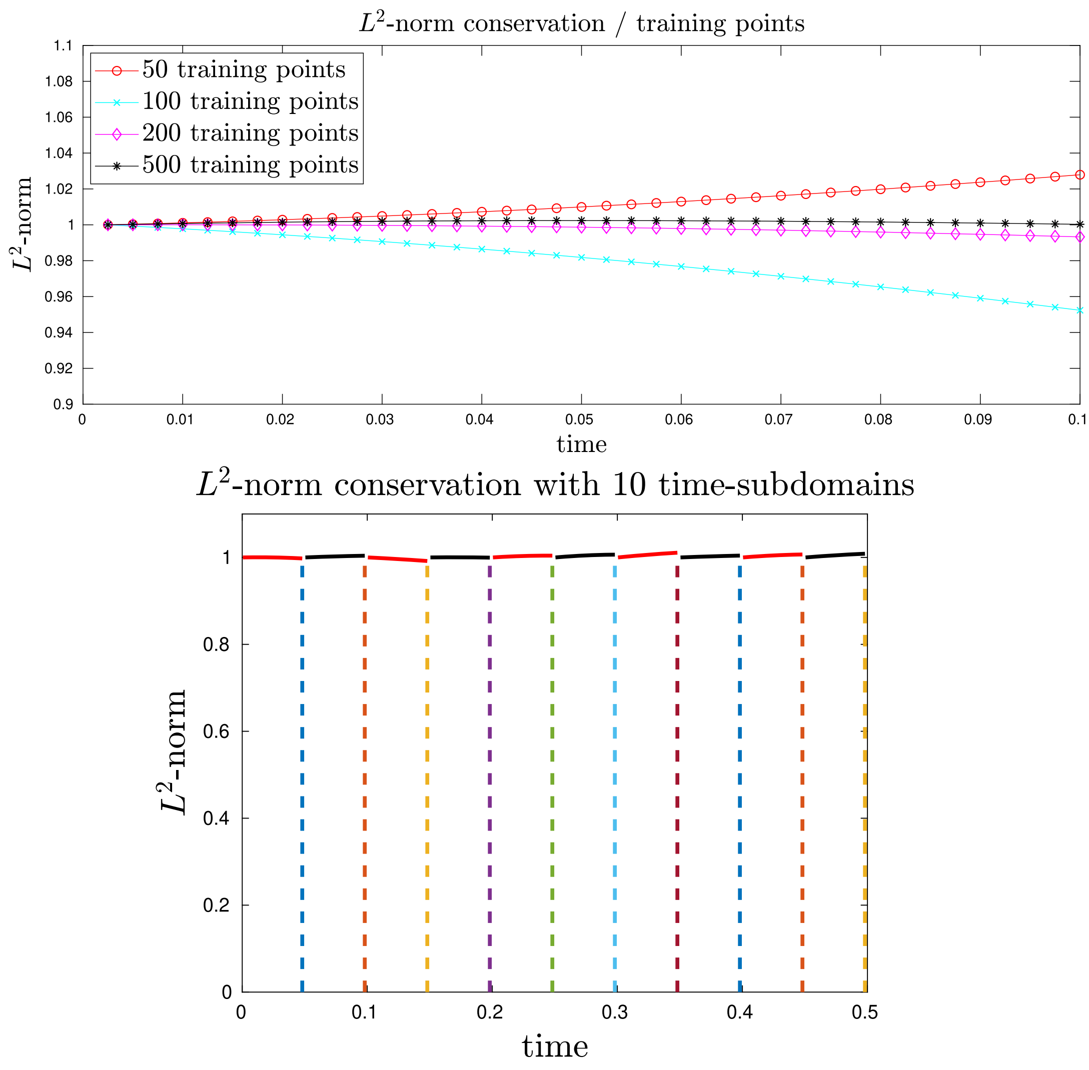


Fig. 3. Experiment 2. (Top) L^2-norm conservation as function of time and with 50, 100, 200, 500 input data: $\{(t, \sum_{i=1}^{4} |N_i(\cdot,t)|^2_{L^2(\Omega);\mathbb{R}}),\ t \in [0, 10^{-1})\}$. (Bottom) L^2-norm conservation as function of time for $\{(t, \sum_{i=1}^{4} |N_i(\cdot,t)|^2_{L^2(\Omega);\mathbb{R}}),\ t \in \cup_{i=0}^{19}[t_i, t_{i+1}] = [0,1]\}$.

Hence, overall in its most simple setting, the PINN algorithm has then a complexity of $O(k_G \mathcal{N}_{\boldsymbol{x}} \mathcal{N}_t \mathcal{N}_{\boldsymbol{\theta}})$. For very large computational times and compared to PDE solvers using fine grids and high order approximations (corresponding to $N_{\boldsymbol{x}} \gg 1$ and ℓ close to 3), the PINN approach may then become competitive. The analysis of accuracy of PINN algorithms is not based on the order of approximation of the space/time derivatives in the Dirac equation and/or on the choice of basis functions. Increasing the precision in the PINN framework has a different meaning and can be performed in different ways. The most natural way to increase the precision is to increase the number of hidden layers and neurons, which is basically equivalent to increase the size of the function set in which we search for the approximate solution. Alternatively or in addition, the optimization solver can be refined/improved by increasing the size of input data set. The latter procedure usually provides a lower local minimum due to the complex structure of the loss function. The latter is naturally one of the main weaknesses of the PINN algorithm.

3.4. Operator-splitting PINN

In this subsection, we discuss an operator splitting-based PINN algorithm. Thanks to the time-domain decomposition, which was discussed above, it is also possible to benefit from a splitting of the Dirac Hamiltonian: the approximate solution to the Dirac equation can be computed thanks to i) PINN for transport part of the Hamiltonian (order 1 operator), and ii) the exact solution to a system of first order differential equations for the mass (algebraic operator), as in [24]. More specifically let us consider:

$$\partial_t \Phi \quad := \quad -H_x \partial_x \Phi - H_y \partial_y \Phi - C\Phi\,, \tag{12}$$

where

$$H_x = c\begin{pmatrix} \sigma_x & 0_2 \\ 0_2 & \sigma_x \end{pmatrix}, \quad H_y = c\begin{pmatrix} 0_2 & -\widetilde{\sigma}_y \\ \widetilde{\sigma}_y & 0_2 \end{pmatrix},$$

and where C is given by

$$\begin{pmatrix} 0_2 & -mc^2\sigma_z \\ mc^2\sigma_z & 0_2 \end{pmatrix}.$$

We first decompose the time-domain as follows: $[0,T]=\cup_{i=0}^{N}[t_i,t_{i+1}]$ with $t_0=0$ and $t_{N+1}=T$ and with $\Delta t=t_{i+1}-t_i$. From time t_i to t_{i+1}, and assuming that a PINN $N(\overline{\boldsymbol{\theta}}_{i+1},\cdot,t_i)$ was computed at time t_i, we compute the PINN at time t_{i+1} in 2 steps:

1. From $[t_i,t^*_{i+1})$, with $t^*_{i+1}=t_i+\Delta t$ we solve

$$\partial_t\Phi \quad := \quad -H_x\partial_x\Phi - H_y\partial_y\Phi\,, \tag{13}$$

   by minimizing the following loss function

$$\overline{\boldsymbol{\theta}}_{i+1} \quad = \quad \mathrm{argmin}_{\boldsymbol{\theta}\in\Theta}\big\|\partial_t\boldsymbol{N}(\boldsymbol{\theta},\cdot,\cdot)+H_x\partial_x\boldsymbol{N}(\boldsymbol{\theta},\cdot,\cdot)+H_y\partial_y\boldsymbol{N}(\boldsymbol{\theta},\cdot,\cdot)\big\|^2_{L^2(\Omega\times[t_i,t_{i+1}))}\,,$$

   which provides an approximation to (13), $\boldsymbol{N}(\overline{\boldsymbol{\theta}}_{i+1},\cdot,t^*_{i+1})$.
2. Then we compute from $[t_i,t_{i+1}]$

$$\partial_t\Phi \quad := \quad -C\Phi\,, \tag{14}$$

   with initial condition $\boldsymbol{N}(\overline{\boldsymbol{\theta}}_{i+1},\cdot,t^*_{i+1})$. This provides

$$\boldsymbol{N}(\overline{\boldsymbol{\theta}}_{i+1},\cdot,t_{i+1}) \quad := \quad \exp(-\Delta tC)\boldsymbol{N}(\overline{\boldsymbol{\theta}}_{i+1},\cdot,t^*_{i+1})\,,$$

   where

$$\exp(-\Delta tC)=\begin{pmatrix} 1 & 1 & e^{mc^2\Delta t} & 1 \\ 1 & 1 & 1 & e^{-mc^2\Delta t} \\ e^{-mc^2\Delta t} & 1 & 1 & 1 \\ 1 & e^{mc^2\Delta t} & 1 & 1 \end{pmatrix}.$$

   Finally $\boldsymbol{N}(\overline{\boldsymbol{\theta}}_{i+1},\cdot,t_{i+1})$ becomes the initial condition from the IBVP (12) for $t\in[t_{i+1},t_{i+2}]$. At each "time-iteration", the splitting produces an error in $O(\Delta t^2[H_x+H_y,C])$.

The interest of using the operator splitting is that the optimization process is only performed for computing the contribution of the transport operators of the Dirac Hamiltonian, while the highly oscillatory contribution ($mc^2\beta$), which is computationally difficult from the optimization point of view, is evaluated "analytically".

Operator splitting approaches have often be used for approximating the solution to the Dirac equation [39,16] using standard PDE solvers. There is here an additional interest in combining operator splitting and NN-based algorithm. As it was described in Subsection 3.2, we can indeed again benefit from a learning from one time iteration to the next, using as ansatz in the optimization algorithm, the converged parameters obtained at the previous iteration. Alternatively, we can even construct neural network for small time interval $[0,\Delta t]$, directly in the form

$$\boldsymbol{N}(\boldsymbol{\theta},\boldsymbol{x},t)\to\boldsymbol{N}(\boldsymbol{\theta},\boldsymbol{x},t)\exp(-\Delta tC)\,.$$

This approach will be explored in future works, with more general operators C of the form

$$C(\boldsymbol{x})=\begin{pmatrix} 0_2 & -V(\boldsymbol{x})I_2-mc^2\sigma_z \\ V(\boldsymbol{x})I_2+mc^2\sigma_z & 0_2 \end{pmatrix}.$$

3.5. Computational error

In the following experiment, we are interested in the behavior of the PINN algorithm when "refining" the PINNs; that is by increasing the number of hidden layers/neurons for a large and fixed input data size. Let us notice that the following test does however not provide or illustrate overall convergence of the PINN algorithm. We refer to [40] for a first rigorous analysis of the PINN convergence. The accuracy of the PINN algorithm largely relies on the efficiency of the optimization algorithm in particular its stochastic version.

Experiment 3. We consider the same setting as in Experiment 2, but on $(-4,4)^2\times[0,T]$, with $\boldsymbol{k}_0=(0,0)^T$ in (7), and $V(\boldsymbol{x})=-1/\sqrt{\|\boldsymbol{x}\|^2+\varepsilon^2}$ with $\varepsilon=0.25$. The objective of this experiment is to illustrate the behavior of the PINN algorithm when the NN is refined. In this goal, we report on Fig. 4 (Left) the $L^2\big((-4,4)^2\big)$-error between the density of reference at $T=2$ computed with 8 hidden layers and 50 neurons, with i) the solutions with 4 hidden layers and 5, 10, 20, 30 neurons, ii) the solutions with 1, 2, 4, 6 hidden layers and 40 neurons. The minimization of the loss function is reported in Fig. 4 (Right). The train (resp. test) loss value is 4.6×10^{-6} (resp. 3.4×10^{-6}). We report in Fig. 5 (Left), the L^2-norm between the density of reference at $T=2$ and the density computed using 1, 2, 4, and 6 hidden layers with 40 neurons. In Fig. 5 (Right.), we report the L^2-norm error between the density of reference at $T=2$ and the density computed on 4 hidden layers and 5, 10, 20, 30 neurons. Both tests illustrate the expected consistency of the overall PINN strategy applied to the TDDE.

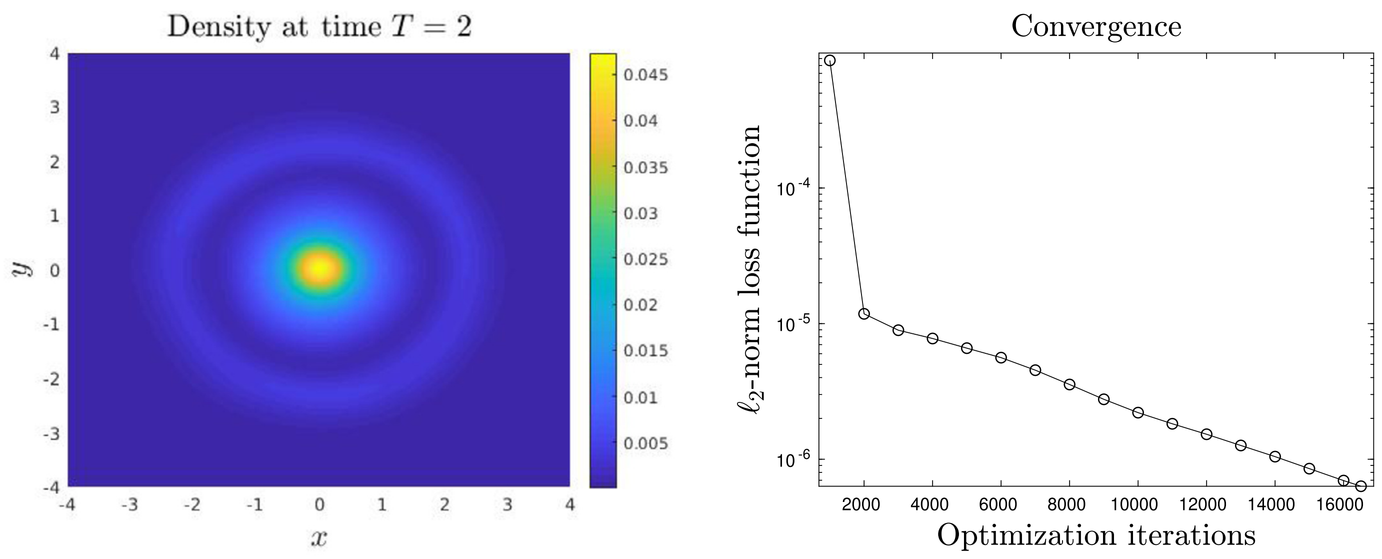


Fig. 4. Experiment 3. (Left) Density at time $T = 2$. (Right) Loss function convergence.

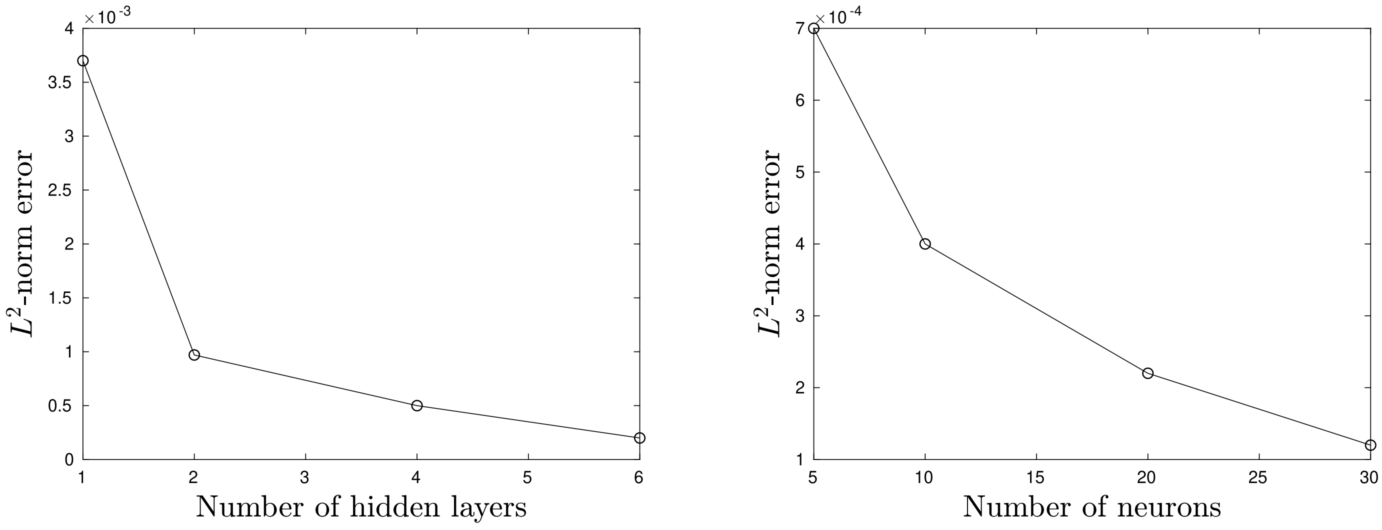


Fig. 5. Experiment 3. L^2-norm "error" between density of reference at $T = 2$. (Left) 1, 2, 4, 6 hidden layers and 40 neurons (Right) 4 hidden layers and 5, 10, 20, 30 neurons.

3.6. Scaling of the PINN solver from the computational point of view for increasing values of c

As discussed above the typical time scale to capture the Zitterbewegung requires time resolution of the order $1/mc^2$; at final time T, the number of input data in the t-direction should scale in Tmc^2. As the PINN-algorithm is linear as a function the number of input data (in any space- or time-direction), see Section 3.3, the complexity of the PINN-algorithm should increase linearly in T and quadratically in c, see Fig. 6 for $(c, T) \in [1, 137] \times [0, 10]$. The analogue requirement for standard numerical TDDE solvers, is the need to take time steps smaller than $1/mc^2$. Notice that different existing Dirac models may involve different values of c: speed of light in quantum relativistic physics, Fermi velocity in graphene modeling, or even $c = 1$ in some other frameworks. This motivates the study of the computational complexity of the PINN algorithm as a function of c.

Experiment 4. It is not simple to numerically reproduce this behavior due to the large number of the parameters in the PINN algorithm. In particular, changing the value of the c, that is changing the value of a physical variables in the Dirac equation, we do not expect similar solutions. We then propose to proceed as follows.

- The following physical data are chosen and fixed: V ($V(\boldsymbol{x}) = -1/\sqrt{\|\boldsymbol{x}\|^2 + 1}$), $m = 1$, space-time domain $(-8, 8) \times (0, 0.25)$; and numerical data: number of hidden layers (5) and neurons (30), learning rate 10^{-4}.
- The following numerical parameters will vary: number of input data and number of training iterations (epoch).

We consider below c equal to 2^i, $i = -1, 0, 1, 2, 3$. We report in the Fig. 7 the train loss functions for the different values of c to show the convergence. This experiment does not allow to rigorously reproduce the theoretical quadratic scaling CPU vs value of c. However it illustrates the rapid nonlinear increase of the computational complexity when the value of c is linearly increased, due to the need to increase the number of input data in t-.

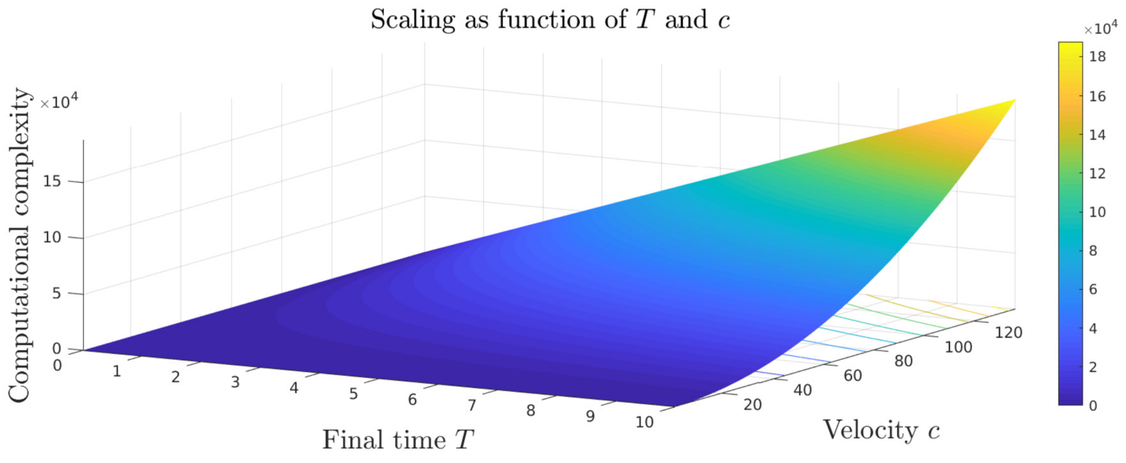


Fig. 6. Scaling of the computational complexity a function of T and c.

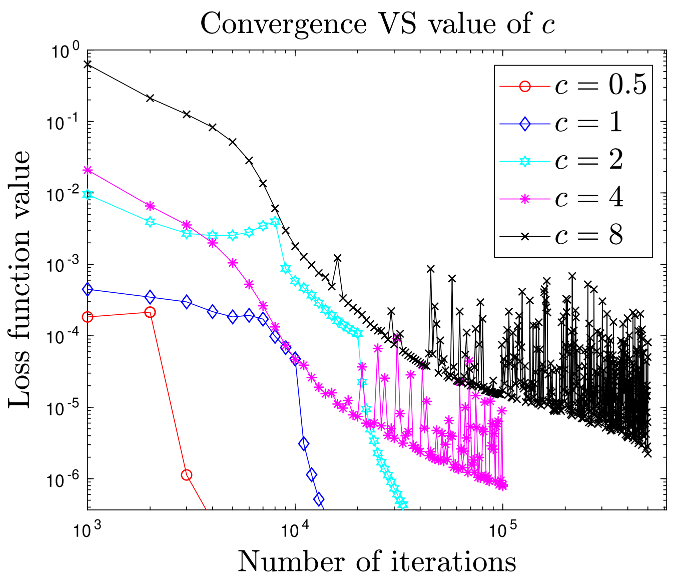


Fig. 7. Experiment 4. Loss function convergence for $c = 2^i$, $i = -1, \cdots, 3$, with the size of the input data set given in Table 1.

Table 1
Number of input data and CPU as function of c.

Value of c	Data set size	CPU time (seconds)
$c = 0.5$	5×10^3	72.2
$c = 1$	7.5×10^3	107.4
$c = 2$	5×10^4	242.6
$c = 4$	5×10^6	1655.7
$c = 8$	10^6	6443.8

3.7. Some physical experiments

In this section we propose some more physics-oriented simulations to illustrate the PINN algorithm.

Experiment 5. We consider the two-dimensional Dirac equation on the interval $(-5, 5)^2 \times [0, 2]$. In this first experiment we take $m = 1$ and $V = 0$. The initial condition is a wavepacket (7) where $\boldsymbol{k}_0 = (1, 1)^T$. The PINN is made of 3 input layers, 8 hidden layers with 40 neurons and 4 output layers. The number of input data is 10^4 for the interior domain and boundary. The number space-time grid points is $75^2 \times 20$ and tanh is chosen as the transfer function. The epoch size is taken equal to 10^3 in the SGD with a learning rate of 10^{-4}. In Fig. 8, we report the density $\boldsymbol{x} \mapsto \rho_N(\boldsymbol{x}, T)$ at time $T = 2$ (Left). At convergence the mean residual (loss function value) is given by 1.12×10^{-4}. The train (resp. test) loss value is 2.2×10^{-7} (resp. 2.7×10^{-7}). We report the loss function ℓ_2-norm as function of optimizer iterations in Fig. 8 (Right), illustrating the convergence of the minimization process.

Experiment 6. The next experiment is in connection with the *Klein paradox* [24,41]. The latter is a phenomenon which occurs in relativistic physics, and which corresponds to planewaves to cross a potential barrier V_0, when the energy E of the incoming wave is lower than

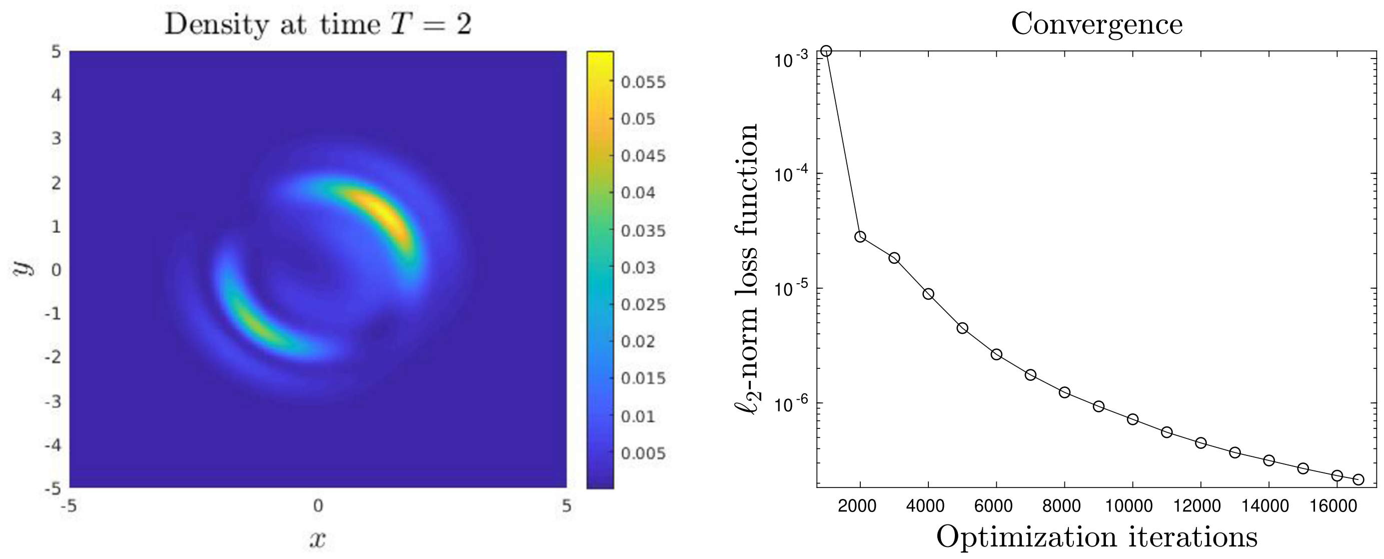


Fig. 8. Experiment 5. (Left) Density at time $T = 2$. (Right) Loss function convergence.

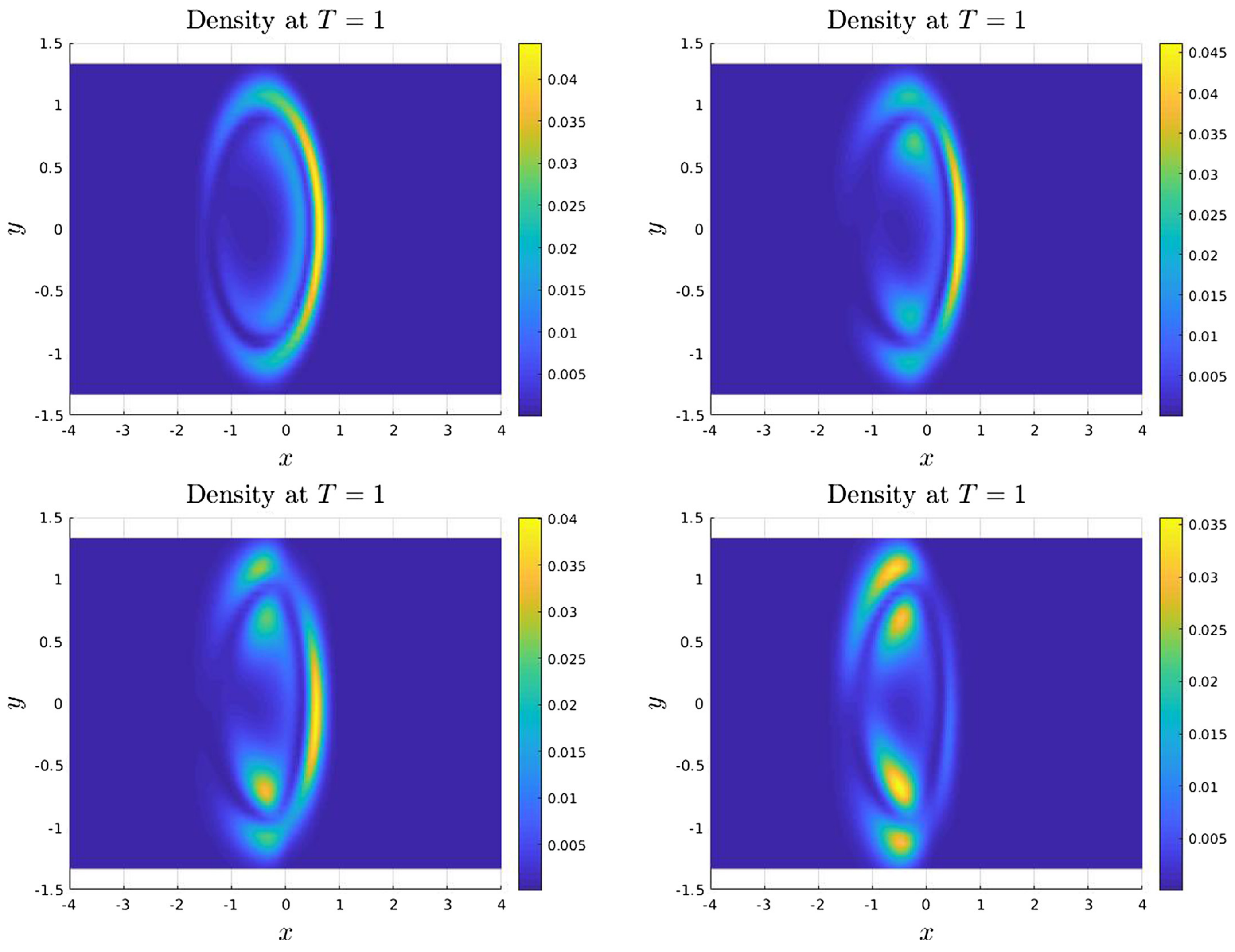
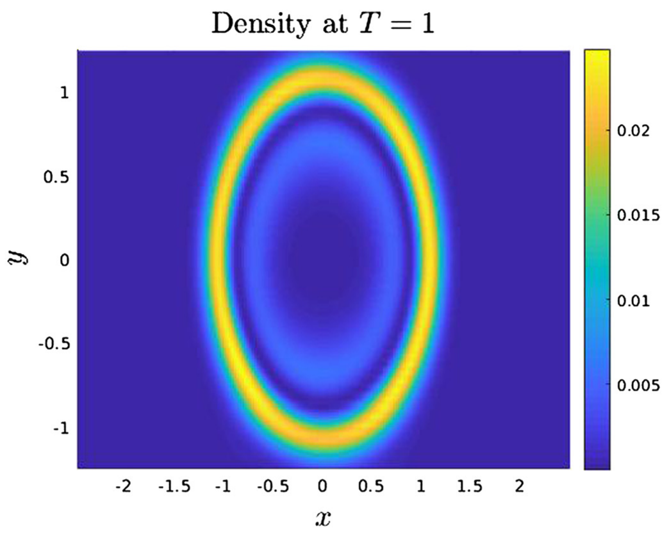


Fig. 9. Experiment 6. From top to bottom: density at $T = 2$ with i) $V_0 = \sqrt{5}$, ii) $V_0 = 2\sqrt{5}$, iii) $V_0 = 3\sqrt{5}$, iv) $V_0 = 4\sqrt{5}$.



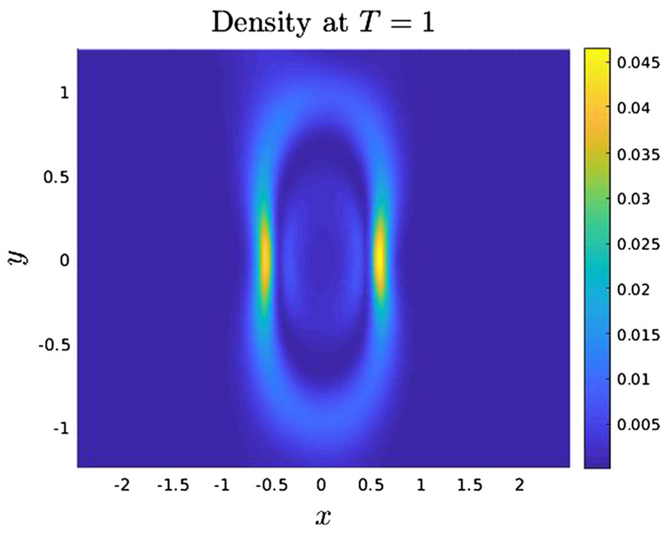


Fig. 10. Experiment 7. (Left) Density at $T = 1$ in flat space. (Right) Density at $T = 1$ in curved space.

the potential. More specifically, wave transmission is possible for $E < V_0 - mc^2$, thanks to the negative energy solution describing anti-fermions, while the incoming and reflected part parts are the electron wavefunction, see [24]. We consider the following potential

$$V(x, y) = \frac{V_0}{2}\Big(1 + \tanh\big(x/L\big)\Big),$$

where $L \ll 1$, and initial condition

$$\phi_0(\boldsymbol{x}) = \exp\big(-20\|\boldsymbol{x} - \boldsymbol{x}_0\|_2^2 + \mathrm{i}\boldsymbol{k}_0 \cdot \boldsymbol{x}\big)(1, C)^T, \tag{15}$$

with $\boldsymbol{k}_0 = (2, 0)$, $C = k_{0;x}/(1 + k_{0;x}^2 + 1)$ and $\boldsymbol{x}_0 = (-1, 0)^T$. The PINN algorithm is applied with 5 hidden layers, 40 neurons. We report in Fig. 9 the density at $T = 0$, then $T = 1$ with $V_0 = 0\sqrt{5}$, $2\sqrt{5}$, and $3\sqrt{5}$, $4\sqrt{5}$. This tests shows that even a part of the wavefunction can still partially cross a potential barrier when $E < V_0 - mc^2$, corresponding to the Klein paradox.

Experiment 7. This last experiment is about the modeling of electron propagation on strained graphene surface. We here illustrate the fact that the implementation of the PINN algorithm is not modified by the “complexification” of the Dirac operator. Basically, we here have $m = 0$, and we arbitrarily take ρ has a Gaussian function loosely corresponding to a Gaussian surface [31], and we neglect the pseudomagnetic field for simplicity [42]. More specifically, we take

$$\rho(\boldsymbol{x}) = 1 + \alpha\exp(-\|\boldsymbol{x} - \boldsymbol{c}_+\|^2) + \alpha\exp(-\|\boldsymbol{x} - \boldsymbol{c}_-\|^2),$$

where $\alpha = 5$, and $\boldsymbol{c}_\pm = (\pm 1, 0)$. Initially, we consider a Gaussian initial wavefunction (7) with $\boldsymbol{k}_0 = (0, 0)^T$. We compare the evolution of the wavefunction in flat and curved spaces, see (10). We observe the focusing effect of the graphene surface deformation, which is coherent with the computations performed in [31]. The changes in the Dirac equation (space-dependent coefficient) do not modify/complexify the PINN algorithm, but naturally modify the complexity for minimizing the loss function.

4. Concluding remarks

In this paper, we have applied the celebrated PINN algorithm to solve the time-dependent Dirac equation in different frameworks, including relativistic quantum physics. Some experiments have been presented to illustrate some mathematical and physical properties when applied to the Dirac equation.

4.1. Summary of the properties

Let us summarize some pros and cons of the PINN algorithm for the Dirac equation, and which were illustrating in this paper.
Pros.

- As any machine learning algorithm, once the network is trained (and the neural network parameters evaluated), the space-time solution can further be evaluated/tested at any space and time locations, without re-computing the full solution.
- The automatic differentiation allows for avoiding some crucial difficulties when approximating the Dirac equation, such as Fermion doubling, numerical diffusion, numerical stability, etc. The latter is usually the source of important difficulties when considering non-constant coefficients in the TDDE.
- The computational method is largely independent of the type of Dirac equation under consideration (mass-free, curved space, relativistic quantum particles), unlike of course the efficiency of the training.
- Learning can be used at different stages of the computation; such as within the time-domain decomposition or operator splitting method.

Cons.

- The main (general) issue with neural network based algorithms is that they usually do not provide a global minimum. Hence there is no guarantee on the accuracy of the computed solution. However, if the network is accurately trained for simple problems (where it is possible to get solutions of references) such networks can be used as initial guess for more complex problems.
- The PINN approach does not allow to circumvent the computational issue coming from the high oscillatory behavior coming from the mass term. In this case, it is necessary to get a NN with fine resolution in-time. However unlike standard computational methods, it does not lead to instability issues.
- By default, the L^2-norm of the wavefunction is not naturally conserved an additional constraints or a time domain decomposition is required for the conservation of the norm for long time propagations.
- Overfitting or underfitting may occur if the training is not properly performed. More generally, the overall accuracy and efficiency largely depend on the minimization algorithm and the structure of the loss function and the dimension of the search space.
- The space of solutions is limited by the structure of neural networks.

4.2. Perspectives

Due to its simplicity this methodology is also a very promising tool for simulating intense laser-molecule interactions in the relativistic regime, which will be the topic of a forthcoming paper.

Acknowledgments. The authors would like to thank Z. Schätzle and G. Carleo for constructive discussions on machine learning algorithms for quantum chemistry. E.L. would also like to thank the Department of Mathematics at UCSB and IPAM (UCLA) for their hospitality and funding. E.L. was partially supported by NSERC through the Discovery Grant program. X.Y. was partially supported by the NSF grants DMS-1818592 and DMS-2109116.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] M. Raissi, P. Perdikaris, G. Karniadakis, J. Comput. Phys. 378 (2019) 686.
[2] G. Pang, L. Lu, G. Karniadakis, SIAM J. Sci. Comput. 41 (2019) A2603.
[3] L. Yang, D. Zhang, G.E. Karniadakis, SIAM J. Sci. Comput. 42 (2020) A292.
[4] A.D. Jagtap, G.E. Karniadakis, Commun. Comput. Phys. 28 (2020) 2002.
[5] I. Lagaris, A. Likas, D. Fotiadis, IEEE Trans. Neural Netw. 9 (1998) 987.
[6] L. Bottou, in: Proceedings of COMPSTAT'2010, Physica-Verlag/Springer, Heidelberg, 2010, pp. 177–186.
[7] S. Sun, Z. Cao, H. Zhu, J. Zhao, IEEE Trans. Cybern. 50 (2020) 3668.
[8] C. Gong, Q. Su, R. Grobe, J. Opt. Soc. Am. B 38 (2021) 3582.
[9] G. Carleo, et al., Rev. Mod. Phys. 91 (2019) 045002.
[10] G. Carleo, M. Troyer, Science 355 (2017) 602.
[11] W. Bao, Y. Cai, J. Yin, SIAM J. Numer. Anal. 59 (2021) 1040.
[12] M. Lewin, É. Séré, Proc. Lond. Math. Soc. (3) 100 (2010) 864.

[13] B. Thaller, The Dirac Equation, Texts and Monographs in Physics, Springer-Verlag, Berlin, 1992.
[14] C. Itzykson, J.B. Zuber, Quantum Field Theory, McGraw-Hill, 1980.
[15] X. Antoine, E. Lorin, Comput. Phys. Commun. 220 (2017) 150.
[16] G.R. Mocken, C.H. Keitel, Comput. Phys. Commun. 178 (2008) 868.
[17] Y. Salamin, S.X. Hu, K.Z. Hatsagortsyan, C.H. Keitel, Phys. Rep. 427 (2006) 41.
[18] F. Fillion-Gourdeau, E. Lorin, A. Bandrauk, J. Comput. Phys. 307 (2016) 122.
[19] I.P. Grant, J. Phys. B, At. Mol. Phys. 19 (1986) 3187.
[20] J.W. Braun, Q. Su, R. Grobe, Phys. Rev. A 59 (1999) 604.
[21] R. Hammer, W. Pötz, A. Arnold, J. Comput. Phys. 256 (2014) 728.
[22] W. Bao, Y. Cai, X. Jia, J. Yin, Sci. China Math. 59 (2016) 1461.
[23] W. Bao, Y. Cai, J. Yin, Math. Comput. 89 (2020) 2141.
[24] F. Fillion-Gourdeau, E. Lorin, A.D. Bandrauk, Comput. Phys. Commun. 183 (2012) 1403.
[25] E. Godlewski, P.-A. Raviart, Numerical Approximation of Hyperbolic Systems of Conservation Laws, Applied Mathematical Sciences, vol. 118, Springer-Verlag, New York, 1996.
[26] B. Després, Analyse numérique et neural networks, Technical report, Université de Paris, 2021, https://www.ljll.math.upmc.fr/despres.
[27] M. Anthony, P.L. Bartlett, Neural Network Learning: Theoretical Foundations, Cambridge University Press, Cambridge, 1999.
[28] H. Robbins, S. Monro, Ann. Math. Stat. 22 (1951) 400.
[29] M. Ramezani Masir, D. Moldovan, F. Peeters, Solid State Commun. 175–176 (2013) 76.
[30] M. Oliva-Leyva, G.G. Naumis, Phys. Lett. A 379 (2015) 2645.
[31] F. Fillion-Gourdeau, E. Lorin, S. MacLean, Phys. Rev. E 103 (2021) 013312.
[32] F. de Juan, M. Sturla, M.A.H. Vozmediano, Phys. Rev. Lett. 108 (2012) 227205.
[33] L. Chai, E. Lorin, X. Yang, Commun. Comput. Phys. (2022).
[34] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, Deepxde: A Deep Learning Library for Solving Differential Equations, 2020.
[35] M. Abadi, et al., TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, Software available from tensorflow.org.
[36] R. Hammer, W. Pötz, A. Arnold, J. Comput. Phys. 265 (2014) 50.
[37] J.C. Strikwerda, Finite Difference Schemes and Partial Differential Equations, second edition, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2004.
[38] A. Bazarkhanova, Physics informed neural networks for solving Dirac equation in $(1+1)$ dimension, Master's thesis, Nazarbayev University, 2021.
[39] F. Fillion-Gourdeau, E. Lorin, A.D. Bandrauk, J. Comput. Phys. 307 (2016) 122.
[40] Y. Shin, Commun. Comput. Phys. (2020) 2042.
[41] F. Sauter, Z. Phys. 69 (1931) 742.
[42] F. Fillion-Gourdeau, E. Lorin, S. Maclean, arXiv:2111.11496, 2021.