J. Plasma Phys. (2022), *vol*. 88, 905880608 © The Author(s), 2022.
Published by Cambridge University Press
This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.
doi:10.1017/S0022377822000939

Unsupervised discovery of nonlinear plasma physics using differentiable kinetic simulations

Archis S. Joglekar [1,2,†] **and Alexander G.R. Thomas** [2]

[1]Ergodic LLC, San Francisco, CA 94117, USA

[2]Department of Nuclear Engineering and Radiological Sciences, University of Michigan, Ann Arbor, MI 48109, USA

(Received 31 May 2022; revised 10 September 2022; accepted 12 September 2022)

Plasma supports collective modes and particle–wave interactions that lead to complex behaviour in, for example, inertial fusion energy applications. While plasma can sometimes be modelled as a charged fluid, a kinetic description is often crucial for studying nonlinear effects in the higher-dimensional momentum–position phase space that describes the full complexity of the plasma dynamics. We create a differentiable solver for the three-dimensional partial-differential equation describing the plasma kinetics and introduce a domain-specific objective function. Using this framework, we perform gradient-based optimization of neural networks that provide forcing function parameters to the differentiable solver given a set of initial conditions. We apply this to an inertial-fusion-relevant configuration and find that the optimization process exploits a novel physical effect.

Key words: plasma nonlinear phenomena, plasma simulation, plasma waves

1. Introduction

Kinetic plasma physics is described, typically, using various formulations or realizations of Boltzmann-like transport equations for the electron and ion species. A special case that is frequently studied is that of the dynamics restricted to one spatial dimension and one momentum dimension ('1D-1V') for a single species of particles, given by

$$\frac{\partial \tilde{f}}{\partial \tilde{t}} + \tilde{v}\frac{\partial \tilde{f}}{\partial \tilde{x}} - \tilde{E}\frac{\partial \tilde{f}}{\partial \tilde{v}} = \left[\frac{\delta \tilde{f}}{\delta \tilde{t}}\right]_{\text{coll}}, \tag{1.1}$$

where the normalized electric field is given by solving $\nabla \cdot \tilde{E} = 1 - \int \tilde{f}\,\mathrm{d}\tilde{v}$, and the normalized particle distribution function is $\tilde{f} = f(t, x, v)/n_e v_{\text{th}}^3$. The quantities are normalized to electrostatic units where $\tilde{v} = v/v_{\text{th}}, \tilde{x} = x/\lambda_D, \tilde{t} = \omega_p t, \tilde{m} = m/m_e$ and $\tilde{E} = eE/m_e v_{\text{th}}\omega_p$. Here, v_{th} is the thermal velocity, ω_p is the plasma frequency, λ_D is the Debye length, and m_e, and e, are the electron mass, and charge, respectively.

† Email address for correspondence: archisj@umich.edu

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

The left-hand side of this equation describes the evolution of the particle distribution in 'macroscopic' field E that arises self-consistently from the one-particle distribution f. The right-hand side provides a description of two-particle and higher correlations (i.e. collisions). Equation (1.1), along with Gauss's law and with a particular form of the collision operator for two-particle interactions, is often termed the Vlasov–Poisson–Fokker–Planck (VPFP) equation set. Solving the VPFP set is often analytically intractable, even in 1D-1V. This is because the left-hand side has a stiff linear transport term, has a nonlinear term in $E\partial f/\partial v$ and can sustain wave propagation and other hyperbolic partial-differential equation (PDE) behaviour. Additionally, the right-hand side is typically represented by a hyperbolic advection–diffusion PDE. Making progress on kinetic plasma physics often requires computational simulation tools.

Numerical solutions to the 1D-1V VPFP equation set have been applied in research on laser–plasma interactions in the context of inertial fusion, for example in plasma-based accelerators (Krall, Joyce & Esarey 1991; Thomas 2016), space physics (Chen, Klein & Howes 2019), fundamental plasma physics (Pezzi *et al.* 2019) and inertial fusion (Strozzi *et al.* 2007; Fahlen *et al.* 2009; Banks *et al.* 2011). Such numerical simulations may be used to explore initial conditions and forcing functions to understand the behaviour of a physical effect in response to input parameters. Multi-dimensional 'brute-force' scans are, however, inefficient and costly and it is therefore beneficial to seek a more guided approach by leveraging optimization techniques.

Optimization techniques can be grouped into gradient-based or gradient-free methods. Gradient-free methods are typically useful when it is not practical to obtain the gradient of the desired objective function (Nocedal & Wright 2006). For example, when the objective function is the outcome of a plasma-based accelerator experiment. In this case, one can use classical approaches like the Nelder–Mead algorithm (Shalloo *et al.* 2020) or more modern alternatives like Bayesian (Shalloo *et al.* 2020; Jalas *et al.* 2021) or evolutionary (He *et al.* 2015; Smith *et al.* 2020) algorithms. These algorithms have been shown to be effective in the small N_p regime, where N_p is the number of parameters to optimize. While there has been some success in using gradient-free approaches in large N_p regimes e.g. in reinforcement learning scenarios (Salimans *et al.* 2017), it has been when the gradient information is unreliable. In general, as $N_p \gg 1$, utilizing gradient-free methods becomes computationally intractable or unstable. In the work here, we learn functions reparametrized via a neural network, and therefore we seek the optimal neural network weights and biases. In this work, $\mathcal{O}(10) \leq N_p \leq \mathcal{O}(10^3)$, and automatic-differentiation-driven, gradient-based methods offer significant advantages.

While not focused on the nonlinear kinetic plasma dynamics, recent works have applied gradient-based optimization to fusion device design. Analytic approaches have resulted in the development of adjoint methods for shape derivatives of functions that depend on magnetohydrodynamic equilibria (Antonsen, Paul & Landreman 2019; Paul *et al.* 2020). These methods have been used to perform optimization of stellarator design (Paul, Landreman & Antonsen 2021). Other work uses gradients obtained from analytic (Zhu *et al.* 2017) and automatic differentiation[1] (AD) (McGreivy, Hudson & Zhu 2021; Conlin *et al.* 2022; Dudt *et al.* 2022; Panici *et al.* 2022) towards similar device optimization goals. Here, we propose the application of gradient-based optimization towards understanding the nonlinear, kinetic plasma dynamics using differentiable kinetic simulations.

Differentiable simulations have been used in a variety of contexts for such guided searches, for example, learning parameters for molecular dynamics (Schoenholz, Cubuk & Jax 2019), learning differencing stencils in PDEs (Bar-Sinai *et al.* 2019; Zhuang *et al.*

[1] See Baydin *et al.* (2018) for a recent machine-learning focused perspective.

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

2020; Kochkov *et al.* 2021) and controlling PDEs (Holl, Koltun & Thuerey 2020). Here, we apply AD towards learning physical relationships and discovering novel phenomena in the VPFP dynamical system by training neural networks through differentiable simulations that solve (1.1).

In the rest of the manuscript, we first develop a conceptual framework for how one might implement gradient-descent-based optimization of parameters and learning of functions via differentiable programs. We then apply this framework towards kinetic plasma physics. Specifically, we extend the findings in Fahlen *et al.* (2009) of the etching of nonlinear plasma wavepackets to study the effect of the hot electrons from one wavepacket on downstream wavepackets using a gradient-based approach. To do this, we train a neural network that provides control parameters to the PDE solver. By choosing physical parameters as inputs and control parameters as outputs of the neural network, we enable the neural network to learn a function that describes the physical relationship between the plasma parameters and the forcing function parameters e.g. the resonance frequency. We train the neural network in an unsupervised fashion using a cost function based on minimizing the free energy and maximizing the non-Maxwellian-ness of the plasma distribution function. This enables us to create self-learning plasma physics simulations, where the optimization process provides a physically interpretable function that can enable physics discovery.

2. Physics Discovery using Differentiable Simulations

In this section, we provide a step-by-step description of how a traditional simulation-based computational physics workflow may be modified to perform closed-loop optimization.

2.1. *Open loop: manual workflow*

Figure 1(*a*) depicts a typical simulation workflow represented as a cyclic graph. The user defines the parametric inputs that create the state vector $\boldsymbol{x}$. This can contain any parameters that are used to define the simulation e.g. the grid size, the number of solver steps, etc. For didactic purposes, the physical parameters to the simulation may be separated from $\boldsymbol{x}$ into a different vector of inputs $\boldsymbol{p_d}$ e.g. the forcing function parameters, the viscosity coefficient etc.

Each of $\boldsymbol{x}$ and $\boldsymbol{p_d}$ is passed to the algorithm that solves the PDE which is represented by the function, $\mathcal{V}$. The output of these simulations is stored in the final state vector $\boldsymbol{x_f}$. The final state is postprocessed using a domain-specific set of algorithms devised by the user or otherwise. The results of the postprocessing are interpreted by the user who then determines the next set of inputs and parameters.

2.2. *Closed loop: brute-force parameter scan*

Figure 1(*b*) shows a more automated workflow. We replace the grey-box postprocessing step with the calculation of a scalar quantity S using a cost function $\mathcal{C}$ on the final state $\boldsymbol{x_f}$. This reduces the complexity of the interpretation of the postprocessing and enables a more rapid search in parameter space. The decrease in required human effort for completing one cycle enables the user to execute this loop as a brute-force parameter scan over a pre-defined parameter space. At the end, the user can look up the minimum/maximum of the scalar cost function, and find the parameters which provide that minimum.

The parameter scan approach scales with the number of different unique parameters and the number of values of each parameter. e.g. a two-dimensional search in x and y requires $N_x \times N_y$ calculations. Therefore, the parameter scan approach quickly becomes inefficient when there are many parameters to scan, or when the required resolution in parameter

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

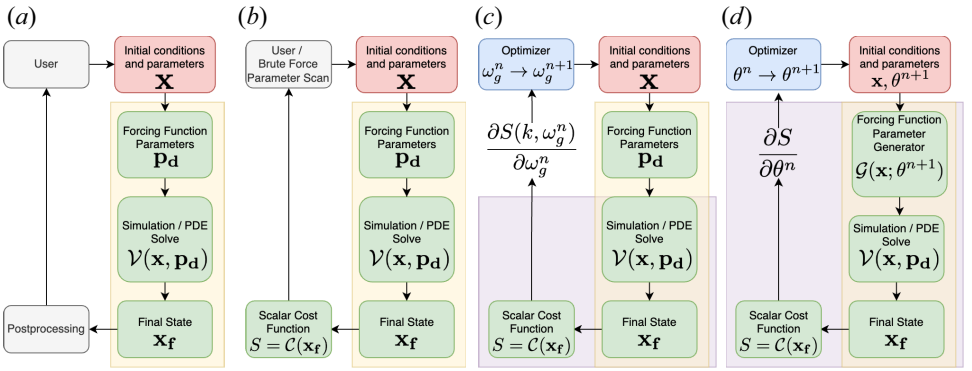


FIGURE 1. (*a*) A typical workflow where the user provides the initial conditions and forcing function parameters to a PDE solve. The output of the solve is stored as the final state $\boldsymbol{x_f}$. The final state is analysed using domain-specific postprocessing algorithms. (*b*) A cost function and a parameter scan are introduced which enables= a closed-loop search. (*c*) A gradient-descent-based optimization algorithm replaces the parameter scan to provide a more efficient search mechanism. This requires the components in the purple background to be written in an auto-differentiable framework. (*d*) We add a neural network that generates the forcing function parameters as a function of other parameters. This generalizes the learnings from (*c*) and enables a continuous representation within the learned parameters.

space is very high. To search this parameter space efficiently, and to escape the linear scaling with each parameter, we can use gradient descent.

2.3. *Gradient-descent-based parameter learning*

Figure 1(*c*) includes two modifications. The user/parameter search grey box has been replaced with a gradient-descent-based optimization algorithm. This algorithm provides the updated parameters, e.g. ω_G, a guess for the resonant frequency of the system, for the next iteration of the loop. The gradient-descent algorithm requires the calculation of an accurate gradient.

Symbolic differentiation is out of the question here as we do not have an analytical form for our system. In the Appendix, we compare the performance of finite differencing to acquire the gradient and confirm that AD is a superior method for this purpose. Therefore, by writing our PDE solver $\mathcal{V}$ and the cost function $\mathcal{C}$ using a numerical framework that supports automatic differentiation, we are able to perform gradient descent. Since

$$\mathcal{S} = \mathcal{C}(\boldsymbol{x}_f) = \mathcal{C}(\mathcal{V}(\boldsymbol{x}, \boldsymbol{p}_d)), \tag{2.1}$$

the gradient for the update step is given by

$$\frac{\partial \mathcal{S}}{\partial \boldsymbol{p}_d} = \frac{\partial \mathcal{C}(\mathcal{V}(\boldsymbol{x}, \boldsymbol{p}_d))}{\partial \boldsymbol{p}_d} = \frac{\partial \mathcal{S}}{\partial \mathcal{V}} \frac{\partial \mathcal{V}}{\partial \boldsymbol{p}_d}. \tag{2.2}$$

For example, if we wish to learn the resonant frequency, ω, that optimizes for the scalar, $\mathcal{S}$, we compute

$$\frac{\partial \mathcal{S}}{\partial \boldsymbol{\omega}} = \frac{\partial \mathcal{S}}{\partial \mathcal{V}} \frac{\partial \mathcal{V}}{\partial \boldsymbol{\omega}}. \tag{2.3}$$

Assuming a well-behaved solution manifold, performing gradient descent tends to reduce the number of iterations required to find the minimum in comparison with an evenly

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

spaced parameter scan, especially when the required resolution is unknown (Nocedal & Wright 2006). While gradient-free methods such as the Bayesian framework, genetic algorithms and others are also often used to perform parametric optimization, we do not perform a more exhaustive comparison because our primary goal is to learn functions, rather than parameters. We discuss this in the following section.

2.4. *Gradient-descent-based function learning*

In the final step, we can replace the lookup-like capability of the parameter optimization and choose to learn a black-box function that can do the same. Through that process, we acquire a continuous representation of the function rather than the discrete version acquired in § 2.3

Here, we choose to use neural networks, with a parameter vector θ, representing the black-box function and providing the fitting function with a large amount of flexibility. This allows us to extend the gradient-descent-based methodology and leverage existing numerical software to implement this differentiable programming loop. Now,

$$\mathcal{S} = \mathcal{C}(\boldsymbol{x}_f) = \mathcal{C}(\mathcal{V}(\boldsymbol{x}, \boldsymbol{p}_d)) = \mathcal{C}(\mathcal{V}(\boldsymbol{x}, \mathcal{G}(\boldsymbol{x}, \boldsymbol{p}_d; \theta)), \tag{2.4}$$

where $\mathcal{G}$ is a function that generates the desired forcing function parameter given a parameter vector $\boldsymbol{\theta}$. To extend the example from § 2.3, ω is now a function given by $\omega = \mathcal{G}(\boldsymbol{x}, \boldsymbol{p}_d; \theta)$.

We compute the same gradient as in (2.3) and add a correction factor that arises because the parameter (vector) is now θ, rather than ω. The necessary gradient for the gradient update is now given by

$$\frac{\partial \mathcal{S}}{\partial \boldsymbol{\theta}} = \left[\frac{\partial \mathcal{S}}{\partial \mathcal{V}} \frac{\partial \mathcal{V}}{\partial \mathcal{G}}\right] \frac{\partial \mathcal{G}(\boldsymbol{x}, \boldsymbol{p}_d; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}. \tag{2.5}$$

Since neural networks typically have $\gg \mathcal{O}(10^2)$ parameters, training these via finite-difference or gradient-free methods is typically avoided except for the most extreme cases (Zhang, Clune & Stanley 2017). It is for this reason that, while it is possible to use finite-difference gradients or even a gradient-free method to perform parameter optimization, these methods are less useful when training neural networks.

3. Discovery of long-lived nonlinear plasma wavepackets

When electrostatic waves are driven to large amplitude, electrons can become trapped in the large potential (Bernstein, Greene & Kruskal 1957; O'Neil 1965). These nonlinear electrostatic wavepackets are dynamically evolving, finite-length analogues of the well-known, time-independent, periodic Bernstein-Greene-Kruskal (BGK) modes described in Bernstein *et al.* (1957). Simulations of stimulated Raman scattering (SRS) in inertial confinement fusion scenarios show that similar large-amplitude waves, but of finite extent, are generated in the laser–plasma interaction, and that particle trapping is correlated with the transition to the high-reflectivity burst regime of SRS (Strozzi *et al.* 2007; Ellis *et al.* 2012).

Simulating wavepackets, similar to those generated in SRS, but in individual electrostatic simulations, has isolated their kinetic dynamics. Fahlen *et al.* (2009) showed in these isolated simulations that resonant electrons transit through the slower moving wavepacket. This is because the resonant electrons have velocity $v \approx v_{ph} = \omega/k$, where ω is the frequency and k is the wavenumber of the wavepacket. On the other hand, the group velocity of the wavepacket is $v_g = \partial\omega/\partial k$. Approximating $\omega = \sqrt{1 + 3k^2}$ gives $v_{ph}/v_g = (3k^2 + 1)/3k^2$. For these wavepackets, this ratio is roughly 3–5. The transit of the

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

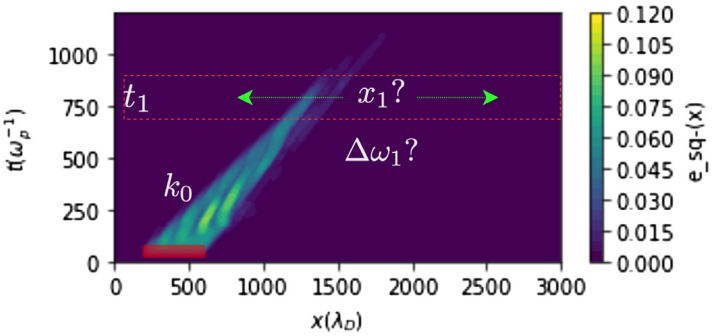


FIGURE 2. Given a first wavepacket with wavenumber k_0 and a desired time of second wavepacket excitation t_1, the task is to learn functions that give the optimal frequency ω_1 and spatial location x_1 of the second wavepacket.

resonant electrons from the back of the wavepacket to the front results in the resumption of Landau damping at the back. The wavepacket is then damped away, as seen in Fahlen *et al.* (2009).

Winjum *et al.* (2019) modelled the interaction of multiple speckles with a magnetic field acting as a control parameter. Since the effect of the magnetic field is to rotate the distribution in velocity space, the field strength serves as a parameter by which the authors control scattered particle propagation. Using this, along with carefully placed laser speckles, they show that scattered light and particles can serve as the trigger for SRS.

Here, we ask: What happens when a nonlinear electron plasma wavepacket is driven on top of another? To answer this question, we reframe it as an optimization problem and ask: What is the best way to excite a wavepacket that interacts with a pre-existing wavepacket?

We start with a large-amplitude, finite-length electrostatic wavepacket driven by a forcing function with parameters given by

$$\boldsymbol{p_0} = [x_0, \omega_0, t_0, k_0]\,, \tag{3.1}$$

where x_i is the location of excitation, ω_i is the frequency, t_i is the time of excitation and k_i is the wavenumber, of the ith wavepacket.

Since we seek to excite a second wavepacket that can interact with the detrapped electrons, we stipulate that the phase velocity of the resonant electrons from both wavepackets is roughly the same. For this reason, we set the wavenumber $k_1 = k_0$. We reparameterize the resonant frequency, ω_1, with a frequency shift, $\Delta\omega_1$ and the linear resonant frequency ω_0 such that $\omega_1 = \omega_0 + \Delta\omega_1$.

We use the time of excitation of the second wavepacket, t_1, as an independent variable along with k_0. For each t_1 and k_0, we seek to learn functions that produce x_1 and $\Delta\omega_1$ i.e. we seek to learn $x_1(t_1, k_0, \omega_0)$ and $\Delta\omega_1(t_1, k_0)$. The entire parameter vector for the second wavepacket is given by

$$\boldsymbol{p_1} = [x_1(t_1, k_0), \Delta\omega_1(t_1, k_0), t_1, k_0]\,. \tag{3.2}$$

This framing is also illustrated in figure 2 where, given k_0 and t_1, we seek functions for ω_1 and x_1.

We reparameterize $\Delta\omega_1$ and x_1 with a neural network with a parameter vector, θ^*, that maximizes the electrostatic energy (minimizes the free energy) and maximizes the

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

difference in the kinetic entropy. These relationships are given by

$$x_1 = x_1(t_1, k_0; \theta^*), \tag{3.3}$$

$$\Delta\omega_1 = \Delta\omega_1(t_1, k_0; \theta^*) \tag{3.4}$$

where[2],

$$\theta^* = \text{argmin}\left[U_{\text{es}}(\boldsymbol{p}; \theta) - \Delta\mathcal{KE}(\boldsymbol{p}; \theta)\right], \tag{3.5}$$

and

$$U_{\text{es}} = \sum_{t_i}^{t_f} \Delta t \sum_x \Delta x E^2 \tag{3.6}$$

$$\Delta\mathcal{KE} = \sum_{t_i}^{t_f} \Delta t \sum_x \Delta x \sum_v \Delta v (f \log(f) - f_{\text{MX}} \log(f_{\text{MX}})), \tag{3.7}$$

are the electrostatic energy and difference in the kinetic entropy terms in the loss function, respectively. Also, $f_{\text{MX}} = f_{\text{MX}}(n(t,x), T(t,x))$, $n(t,x) = \int f(t,x,v)\,\mathrm{d}v$, $T = \int f(t,x,v)v^2\,\mathrm{d}v/n$, where f_{MX} is the local Maxwell–Boltzmann distribution. Equation (3.7) describes the deviation of the local distribution function from the equivalent Maxwell–Boltzmann distribution function. It has been used as a measure of the non-Maxwellian-ness by Kaufmann & Paterson (2009). The reason we maximize this deviation is because we seek to enhance the likelihood of nonlinear kinetic effects.

We vary the independent variables such that

$$k_0 \in [0.26, 0.27, \ldots, 0.32], \tag{3.8}$$

$$t_1 \in [400, 500, \ldots, 800], \tag{3.9}$$

giving an input space of 35 samples from which we seek to learn these functions.

We are able to reproduce similar model training results using a few different neural network configurations and learning parameters. To summarize that study, we find vanilla multi-layer perceptrons 8 nodes wide and 2 layers deep to be effective, especially when 'activated' using the Leaky-Rectified-Linear-Unit function. The specifics behind the neural network, optimizer, and data normalization parameters are provided in the Appendix.

The training simulations are performed with the following parameters. The grid is discretized using $N_x = 6656, N_v = 160, N_t = 1200, t_{\max} = 1100\omega_p^{-1}, x_{\max} = 6600\lambda_D$. Assuming inertial fusion conditions where $n_e = 10^{20}\,\text{cm}^{-3}$, $T_e = 2\,\text{keV}$ gives a non-zero collision frequency $\nu_{ee}/\omega_p = 5 \times 10^{-5}$ (Huba 2011) for the simulations shown here. Details on the solvers and forcing function parameters are provided in the Appendix.

Figure 3 shows that the loss value is reduced over the duration of the training process. Each simulation takes approximately 3 minutes to run on a NVIDIA T4 GPU. We train for 60 epochs. The convergence in the loss metric suggests that we were able to train a overparametrized neural network with 35 samples of data in 60 epochs. While Metz *et al.* (2022) discuss that differentiating through the evolution of chaotic dynamical systems can be ill posed, ablation studies indicate that the objective function posed here has a robust minimum. One reason is because the chaotic particle dynamics is subject to dispersion,

[2]As is convention, we normalize the inputs and outputs to the neural network. Appendix C provides the specific values and methodology used here.

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

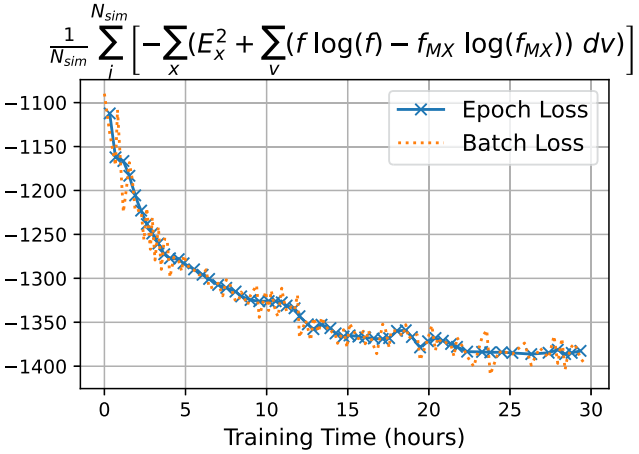


FIGURE 3. The loss, a sum of (3.6) and (3.7), is plotted as a function of time. Each cross represents an epoch, and batch-wise fluctuations are also displayed. The training converges after roughly 150 GPU hours on a NVIDIA T4 and 2100 simulations, which amounts to 60 epochs.

Landau damping and collisions. This may result in a smoothing of the loss surface. Second, since the objective functions used here are space, time and velocity integrals over the chaotic trajectories, the gradients calculated here may benefit from the smoothness associated with time- and phase-space averaging of a well-behaved ergodic system.

Figure 4 shows the electric field profile for three different simulations. In figure 4(a), only the first wavepacket is excited, and in figure 4(b), only the second wavepacket is excited. In figure 4(c), both wavepackets are excited. Early in time, $t = 400\omega_p^{-1}$ (green), when only the first wavepacket has been excited figures 4(a) and 4(c) agree perfectly. The second wavepacket is excited at $t = 500\omega_p^{-1}$. At $t = 900\omega_p^{-1}$, once some time has passed after the excitation of the second wavepacket, the first wavepacket has not fully damped away. It is visible as small bumps in figures 4(a) and 4(c). The second wavepacket is also present at this time and easily seen in figure 4(b). A larger-amplitude wavepacket is seen in figure 4(c). Late in time, the difference in amplitude between the second wavepacket in figures 4(b) and 4(c) is obvious. The second wavepacket has nearly damped away in figure 4(b). In figure 4(c), the second wavepacket persists, at nearly the same energy as it was at $t = 900\omega_p^{-1}$. We observe this superadditive behaviour, where $f(x) + f(y) \leq f(x + y)$, for all wavenumbers we model.

To determine the mechanism behind this phenomenon, we turn to the phase-space dynamics. In figure 5, (i) is a space–time plot of the electric field. The two dashed-dot red lines at the front and back of the wavepacket are parallel and indicate the velocity of the wavefront. In figure 5(a)(i), the rear of the wavepacket propagates at a seemingly faster rate than the front. Fahlen *et al.* (2009) describe this as etching of the wavepacket. In figure 5(b), the wave survives for a much longer time, as was also illustrated in figure 4. Figure 5(ii, iii) are phase-space plots with their centre indicated by the intersection of the horizontal timestamp line and the dashed-dot line at the rear wavepacket. The insets (iv) and (v) correspond to the intersection with the dashed-dot line at the front. Insets (ii) and (iv) show the phase space within a window in $\hat{x}$, while (iii) and (v) are the spatially averaged distribution function. Insets (iii) and (v) serve as a proxy for approximating the propensity of Landau damping in that region. In figure 5(a)(ii, iii), we see that the rear of the wavepacket is Maxwellian. Fahlen *et al.* (2009) showed this is why the rear of the wavepacket damps faster than the front, as in figure 5(a)(i).

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

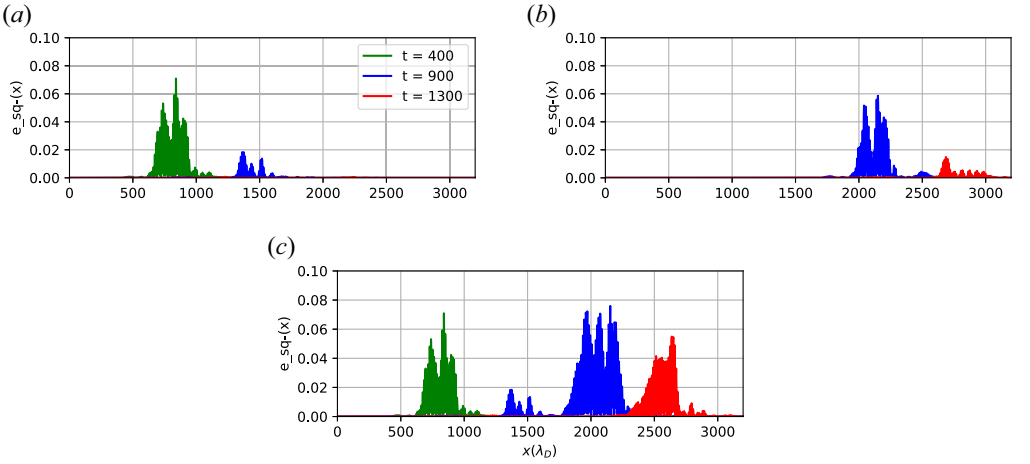


FIGURE 4. Early in time (green), (*c*) is the same as (*a*). Later in time, $t = 900\omega_p^{-1}$ (blue), (*a*,*c*) show very similar magnitudes for the first wavepacket near $x = 1500\lambda_D$ but the second wavepacket excitation is larger in (*c*) than (*b*). At $t = 1300\omega_p^{-1}$ (red), it is clear that (*c*) is not a superposition of (*a*,*b*) because (*b*) has damped away, while (*c*) retains electrostatic energy, suggesting the involvement of a superadditive process. (*a*) First wavepacket only; (*b*) second wavepacket only; (*c*) both wavepackets.

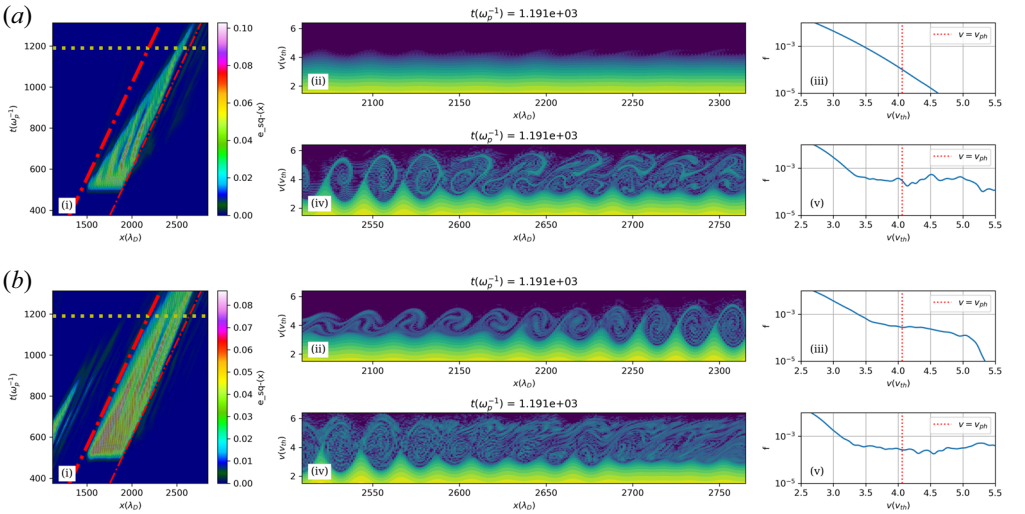


FIGURE 5. Left – space–time plot of the electrostatic energy shows the long-lived wavepacket in (*b*) where the field in (*b*) survives for a longer duration than in (*a*). The horizontal line indicates the timestamp of the snapshots in the middle and right. The diagonal dashed-dot lines indicate the spatial location of the snapshots in the middle and right. Middle phase-space plots at the back (top) and front (bottom) of the wavepacket. In (*b*), the phase space shows significant activity at the back of the wavepacket while in (*a*), the distribution function is nearly undisturbed. Right – T = the spatially averaged distribution function. This confirms the fact that the distribution function has returned to a Maxwell–Boltzmann at the back of the wavepacket in (*a*), while in (*b*), the distribution function remains flat at the phase velocity of the wave. This is the reason behind the loss of damping. (*a*) Second wavepacket only; (*b*) both wavepackets.

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

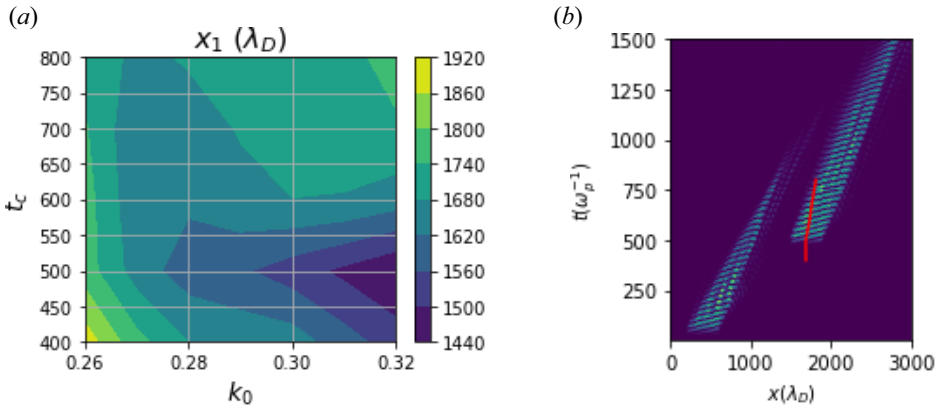


FIGURE 6. (*a*) Learned function for the resonant spatial location as a function of wavenumber of the first wavepacket and time of excitation of the second. (*b*) The locus in space–time (in red) where long-lived wavepackets can be excited for $k = 0.28$: (*a*) $x_1(k_0, t_1)$; (*b*) space–timelocus.

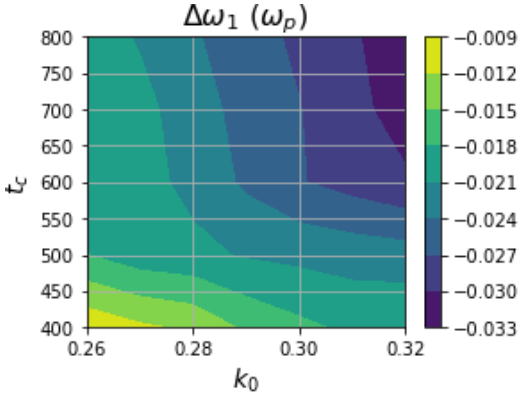


FIGURE 7. The learned function for the frequency shift $\Delta\omega_1(k_0, t_1)$ as a function of wavenumber of the first wavepacket, k_0, and time of excitation, t_1, of the second.

In the simulations described here, the plots in figure 5(*b*) show that the distribution function at the back of the wavepacket has trapped particle activity (figure 5(*b*)(ii)) and there is a near zero slope at the phase velocity of the wave (figure 5(*b*)(iii)). Both plots show that the slope is negligible because of the arrival of streaming detrapped particles from the first wavepacket. Due to this effect, the re-emergence of Landau damping that occurs due to the loss of trapped particles in isolated wavepackets no longer occurs here. This results in a reduction of the etching and the wavepacket propagates freely for some time while the particles from the first wavepacket propagate and arrive at the rear of the second wavepacket.

Figure 6 shows the results of the optimization process for the resonant spatial location. For $t_1 < 500\omega_p^{-1}$, the resonant location decreases as a function of wavenumber. From analysing phase space, we have determined that the spatial location is related to the resonant electron transport i.e. $x_1 \approx v_{ph}t_1$. Waves with larger wavenumbers have smaller phase velocities. Because of this, the resonant spatial location decreases as a function of wavenumber. For a fixed wavenumber, figure 6(*b*) shows a locus of points in space–time where long-lived wavepackets can be excited in the presence of a pre-existing wavepacket.

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

This suggests the possibility of a critical space–time radius within which collisional relaxation has yet to occur, and nonlinear effects can be exploited.

Likewise, we also learn the dependence of the optimum frequency shift as a function of k_0 and t_c, as shown in figure 7. The learned frequency shift, a few per cent here, is similar in magnitude as to that observed in previous work related to SRS (Ellis *et al.* 2012). Furthermore, the frequency shift increases in magnitude as a function of wavenumber. As before, waves with larger wavenumbers have smaller phase velocities, and therefore, interact with more particles. Because of this, waves with larger wavenumbers have a larger nonlinear frequency shift associated with them, as we see here (Manheimer & Flynn 1971; Dewar 1972; Morales & O'Neil 1972; Berger *et al.* 2013).

4. Conclusion

We show how one may be able to discover novel physics using differentiable simulations by posing a physical question as an optimization problem. This required domain expertise in determining which functional dependencies to learn using neural networks.

In figure 1, we show how one may adapt an existing computational science workflow to the autodidactic process described here. In the work performed here, this process enabled the discovery of functional relationships between optimal parameters in a four-dimensional search space with known bounds but an unknown resolution requirement. We trained the model over a coarse grid in k_0 and t_1, and learned functions for x_1 and ω_1. Using gradient descent here allows an escape from the curse of dimensionality and reduces the problem from a four-dimensional search to a two-dimensional search + two-dimensional gradient descent.

This discovery process is not limited to differentiable simulations. While in figure 1, $\mathcal{V}$ represents a PDE solve, it only needs to be a AD-enabled function that is a model for a physical system. For example, rather than a PDE solve, $\mathcal{V}$ could represent a pre-trained neural-network-based emulator for experimental data. In such a scenario, one may be able to learn forcing function parameters for an experiment using the proposed workflow.

In the neural network literature, the gradient required for the update is $\partial\mathcal{S}/\partial\theta = \partial\mathcal{S}/\partial\mathcal{G} \times \partial\mathcal{G}/\partial\theta$. We see that this is the same as (2.5) after the addition of one more node in the computational graph for $\mathcal{V}$, the function that models the physical system. This allows the neural network training process to become unsupervised and data efficient.

It remains to be determined how the physical mechanism discussed here behaves over a range of amplitudes and collision frequencies in addition to the wavenumber scan we perform here. Reduced models of the wavepacket dynamics in SRS remain useful for the development of inertial confinement fusion schemes where laser–plasma instabilities occur.

Differentiating through an entire simulation, in this case, required the intermediate storage of all $N_t \times N_x \times N_v$ arrays representing the distribution function in time and phase space. To enable this on a single NVIDIA T4 GPU, we use gradient checkpointing (Griewank & Walther 2000; Chen *et al.* 2016), as implemented in JAX, at every timestep. That is, we require that the Jacobian of the function representing a single timestep recompute its internal linearization points rather than save them to memory during the forward pass. Another technique by which we can save on the memory cost is by improving the efficiency of the accumulation of the Jacobian e.g. as proposed in Naumann (2004) and similar works. An example application towards a PDE is provided by Oktay *et al.* (2021). A final route to memory savings we wish to highlight here is by solving a backwards-in-time ordinary-differential equation for the adjoint (Pontryagin 1987; Chen *et al.* 2018). For modelling systems of higher-dimensionality, larger time scales and other constraints that

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

require larger in-memory calculations, it will be crucial to leverage methods that help circumvent the memory burden encountered with differentiable simulations.

Acknowledgements

A.J. wishes to acknowledge D. Strozzi for early and continued encouragement, S. Hoyer for introducing him to JAX, M. Poli and S. Masseroli for the perspectives of ML researchers and B.B. Afeyan, W.B. Mori and B.J. Winjum for discussions on nonlinear electron plasma waves. A.J. also gratefully acknowledges travel support from Syntensor Inc. The authors thank the anonymous reviewers for valuable feedback.

Editor William Dorland thanks the referees for their advice in evaluating this article.

Declaration of interests

The authors report no conflict of interest.

Appendix A. Simulation details

We reproduce the solvers in Joglekar & Levy (2020) using JAX (Bradbury *et al.* 2018) and Haiku (Hennigan *et al.* 2020) to allow the usage of AD, judicious placement of neural networks as well as the ability to run on GPU.

A.1. *Solvers*

A.1.1. *Vlasov equation*

We discretize $f(t, x, v)$ with $f^n(x_i, v_j)$. To calculate f^{n+1}, we use the sixth-order Hamiltonian integrator introduced in Casas *et al.* (2017).

As in Joglekar & Levy (2020) and Thomas (2016), the individual components of the left-hand side of (1.1) are solved using operator splitting, spectral discretizations and exponential integration as given by

$$f^{n+1}_{v\partial_x f}(x_i, v_j) = \mathcal{F}_x^{-1}\left[\mathcal{F}_x\left(f^n(x_i, v_j)\right)\exp(-\mathrm{i}k_x(x_i)v_j\Delta t)\right] \tag{A1}$$

$$f^{n+1}_{E\partial_v f}(x_i, v_j) = \mathcal{F}_v^{-1}\left[\mathcal{F}_v\left(f^n(x_i, v_j)\right)\exp(-\mathrm{i}k_v(v_j)E(x_i)\Delta t)\right], \tag{A2}$$

where $\mathcal{F}_{\{x,v\}}$ are the Fourier transforms in space and time, respectively. We use a spectral solver for Gauss's law such that

$$E^n = \mathcal{F}_x^{-1}\left[1 - \frac{\mathcal{F}_x\left(\sum f^n(x_i, v_j)\Delta v\right)}{\mathrm{i}k_x}\right]. \tag{A3}$$

A.1.2. *Fokker–Planck Equation*

Two simplified versions of the full Fokker–Planck operator are implemented. The first of these implementations is given in Lenard & Bernstein (1958) and has the governing equation given by

$$\left(\frac{\delta f}{\delta t}\right)_{\mathrm{coll}} = \nu\frac{\partial}{\partial v}\left(vf + v_0^2\frac{\partial f}{\partial v}\right), \tag{A4}$$

where

$$v_0^2 = \int v^2 f(x, v)\,\mathrm{d}v, \tag{A5}$$

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

is the thermal velocity of the distribution. We term this the LB operator as per the authors' names. The second of these implementations is given in Dougherty (1964) and has a governing equation given by

$$\left(\frac{\delta f}{\delta t}\right)_{\text{coll}} = \nu \frac{\partial}{\partial v}\left((v - \underline{v})f + v_t^2 \frac{\partial f}{\partial v}\right), \tag{A6}$$

where

$$\underline{v} = \int v f(x, v)\, \mathrm{d}v, \tag{A7}$$

is the mean velocity of the distribution and

$$v_t^2 = \int (v - \underline{v})^2 f(x, v)\, \mathrm{d}v, \tag{A8}$$

is the thermal velocity of the distribution while accounting for the mean velocity. We term this the DG operator as per the author's name. The DG operator extends the LB operator by enabling momentum conservation for distribution functions that have a non-zero mean velocity.

We discretize these equations using a backward Euler method with centre differencing space. This procedure results in the timestep scheme given by

$$f^n = \left[\text{LD} \times (\bar{v}_{j+1} f_{j+1}^{n+1}) + \text{DI} \times (f_j^{n+1}) + \text{UD} \times (\bar{v}_{j-1} f_{j-1}^{n+1})\right], \tag{A9}$$

where

$$\text{LD} = \Delta t \nu \left(-\frac{v_{0,t}^2}{\Delta v^2} - \frac{1}{2\Delta v}\right) \tag{A10}$$

$$\text{DI} = \left(1 + 2\Delta t \nu \frac{v_{0,t}^2}{\Delta v^2}\right) \tag{A11}$$

$$\text{UD} = \Delta t \nu \left(-\frac{v_{0,t}^2}{\Delta v^2} + \frac{1}{2\Delta v}\right), \tag{A12}$$

where $\bar{v} = v$ or $\bar{v} = v - \underline{v}$ depending on the implementation. This forms a tridiagonal system of equations that can be directly inverted.

A.2. *Parameterizing the Ponderomotive Driver*

Similar to that presented by Afeyan *et al.* (2014), the ponderomotive driver is parametrized using space–time envelopes created using hyperbolic tangents given by

$$g(p_{i,s}, \boldsymbol{s}) = 0.5\left(\tanh\left[\frac{\boldsymbol{s} - p_{L,s}}{p_{r,s}}\right] - \tanh\left[\frac{\boldsymbol{s} - p_{R,s}}{p_{r,s}}\right]\right), \tag{A13}$$

where $\boldsymbol{s}$ can be $\boldsymbol{x}$ or $\boldsymbol{t}$, $p_{R/L} = p_c \pm p_w/2$, and the rest of the parameters are given by the set $p_i = (p_c, p_w, p_r, k, a, \omega, \Delta\omega)$. Using (A13), the overall profile in space and time is given by

$$\sum_i \Delta E(p_i) = \sum_i \left[g_i(\boldsymbol{t}) g_i(\boldsymbol{x})\; k_i\; a_i\;\; \sin(k_i \boldsymbol{x} - (\omega_i + \Delta\omega_i)\boldsymbol{t})\right]. \tag{A14}$$

In the simulations performed in this work, we specify the following.

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

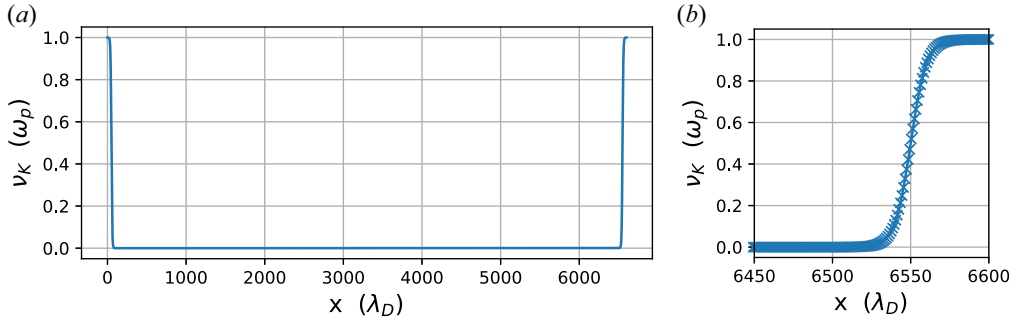


FIGURE 8. We implement thermal-bath boundaries for the plasma by artificially increasing the collision frequency of the Krook operator at the edges using (A13) and $p_L = 500$, $p_R = 6000$, $p_{wL} = p_{wR} = 500$. (*a*) Entire box; (*b*) zoom in.

(i) For each individual set of driver parameters, $p_{r,\boldsymbol{x}}/\lambda_D = p_{r,\boldsymbol{t}}/\omega_p^{-1} = p_r = 10$, $p_{w,\boldsymbol{x}} = 400\lambda_D$ and $p_{w,\boldsymbol{t}} = 50\omega_p^{-1}$.
(ii) For the first driver, $p_{c,\boldsymbol{t}} = 50\omega_p^{-1}$, $p_{c,\boldsymbol{x}} = 400\lambda_D$.

This leaves p_c for the second driver and the wave parameters, k, a, ω, $\Delta\omega$ to be varied in this work. For readability, in the manuscript body, we use $p_{i,c,\boldsymbol{x}} = x_i$ and $p_{i,c,\boldsymbol{t}} = t_i$.

A.3. *Physical domain*

We initialize the plasma distribution function with a Maxwell–Boltzmann distribution with $n_0 = 1.0$, $v_{\text{th}0} = 1.0$, $f(t = 0, x, v) = \exp(-v^2)/\sqrt{\pi}$.

In order to ensure that there is no unwanted interference from particles transiting across the periodic boundary, thermal boundaries are implemented using a Krook-type collision operator (Bhatnagar, Gross & Krook 1954). The equation is solved directly and exactly, and the solution at the new timestep is given by

$$f^{n+1} = f^n(1 - \exp(\nu_K \Delta t)) + f^n_{\text{MX}} \exp(\nu_K \Delta t), \tag{A15}$$

where $f_{\text{MX}}(n_0, v_{\text{th}0}) = n_0 \exp(-v^2/v_{\text{th}}^2)/\sqrt{\pi v_{\text{th}}^2}$.

By having non-zero ν_K only in a small localized region at the boundary, as also described by Strozzi *et al.* (2007), and as shown in figure 8, the particles are absorbed in a fixed Maxwellian, similar to having a thermal boundary condition. This localization is also performed using the hyperbolic tangent parameterization from (A13) but using $1 - g(p, \boldsymbol{x})$. In this case, $p_c = 3300$, $p_w = 6500$, $p_r = 10$.

Appendix B. Neural network and optimizer details

We implement our solver framework using Haiku (Hennigan *et al.* 2020). In this work, we leverage a vanilla multi-layer perceptron constructed using linear (fully connected) layers and nonlinear (activation) functions. We train this model with the following architectures:

$$\mathcal{I} \to \mathcal{L}(4) \to \mathcal{R} \to \mathcal{L}(4) \to \mathcal{R} \to \mathcal{L}(\text{len}(O)) \to \mathcal{T}, \tag{B1}$$
$$\mathcal{I} \to \mathcal{L}(4) \to \mathcal{K} \to \mathcal{L}(4) \to \mathcal{K} \to \mathcal{L}(\text{len}(O)) \to \mathcal{T}, \tag{B2}$$
$$\mathcal{I} \to \mathcal{L}(8) \to \mathcal{R} \to \mathcal{L}(8) \to \mathcal{R} \to \mathcal{L}(\text{len}(O)) \to \mathcal{T}, \tag{B3}$$
$$\mathcal{I} \to \mathcal{L}(8) \to \mathcal{K} \to \mathcal{L}(8) \to \mathcal{K} \to \mathcal{L}(\text{len}(O)) \to \mathcal{T}, \tag{B4}$$

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

where $\mathcal{L}(N)$ is the linear layer with N nodes,

$$\mathcal{R}(x) = \begin{cases} 0.0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}, \tag{B5}$$

$$\mathcal{K}(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}, \tag{B6}$$

and $\mathcal{T}(x) = \tanh(x)$. Also, $\mathcal{I}$ and $\mathcal{O}$ are the arrays representing the inputs to and outputs from the neural network. We observe that the higher capacity architectures we tested converge faster as does training with the Leaky-ReLU function. We use the ADAM optimizer (Kingma & Ba 2017) implemented via Optax (Babuschkin *et al.* 2020). We train the model using learning rates of 0.05, 0.01, 0.005 and we find that the smallest learning rate performs the best. We do not perform extensive architecture and hyperparameter searches because the primary goal here is to discuss and apply the capability of function approximation towards physics discovery, rather than learning the best possible approximation.

Appendix C. Data normalization

Much like how simulations are generally conducted in normalized units, neural networks are typically trained on normalized data. We need to do the same here so we normalize the inputs between 0 and 1. As specified in the body of the text, our inputs to and outputs from the neural network are given by

$$x_1 = x_1(t_1, k_0; \theta^*), \tag{C1}$$

$$\Delta\omega_1 = \Delta\omega_1(t_1, k_0; \theta^*). \tag{C2}$$

We augment the input vector with ω_0, k_1, ω_1 and normalize the inputs to and outputs of the neural network such that

$$x_1 = x_n \times \tilde{x}_1(\tilde{t}_1, \tilde{k}_0, \tilde{\omega}_0, \tilde{k}_1, \tilde{\omega}_1; \theta^*) + x_s, \tag{C3}$$

$$\Delta\omega_1 = \omega_n \times \widetilde{\Delta\omega}_1(\tilde{t}_1, \tilde{k}_0, \tilde{\omega}_0, \tilde{k}_1, \tilde{\omega}_1; \theta^*) + \omega_s. \tag{C4}$$

It is important to note that, since $k_0 = k_1, \omega_1 = \omega_0$ and that $\omega = \omega(k)$ through the electrostatic dispersion relation, there is no new information being added to the training process by this augmentation. However, we do provide the training process with a sophisticated reinterpretation of the existing information by providing $\omega = \omega(k)$.

We normalize the 5 input parameters to scale between 0 and 1 such that $0.26 < k < 0.34$, $1.05 < \omega < 1.3$ and $400 < t_1 < 800$ i.e. $\tilde{k} = (k - k_{\min})/(k_{\max} - k_{\min})$ and so on.

The output parameters, $\tilde{x}_1, \widetilde{\Delta\omega}_1$, are renormalized slightly differently because the outputs range from -1 to 1. For x_1, we allow the entire domain except for regions near the boundaries. We choose the bounds of the frequency shift by measuring typical nonlinear frequency shifts observed for idealized, periodic nonlinear electron plasma waves. We normalize each such that $500 < x_1 < 6000$ and $-0.06 < \Delta\omega_1 < 0.06$. This gives $x_n = 2750, x_s = 3250$ and $\omega_n = 0.06, \omega_s = 0$.

Appendix D. Validation tests

We reproduce the tests implemented and shown in Joglekar & Levy (2020).

(i) Gauss's law.

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

(ii) Landau damping.
(iii) Density conservation of both implementations of the Fokker–Planck operators.
(iv) Momentum conservation of the DG implementation of the Fokker–Planck implementation.
(v) Energy conservation of the LB and DG implementations of the Fokker–Planck operator.

D.1. *Validating the differentiable simulator by recovering known physics*

We describe an additional test here which involves recovering the linear, small-amplitude Langmuir, or electrostatic, resonance using a gradient-based method enabled by this AD-capable implementation. This ensures that the gradients given by the AD system are representative of physical phenomenon.

D.1.1. *Electrostatic waves*

A fundamental wave in plasma physics is the electrostatic wave in unmagnetized plasmas. The dispersion relation is given in numerous textbooks, and reproduced here as

$$1 + \frac{\omega_e^2}{k^2} \int \mathrm{d}v \frac{\mathrm{d}g(v)/\mathrm{d}v}{\omega - kv} = 0, \tag{D1}$$

where ω_e is the plasma frequency, k is the wavenumber, ω is the resonant frequency and v is the independent variable representing the velocity in the integral. Also, $g(v)$ is the normalized distribution function of the plasma particles. This equation has been solved numerically and a lookup table for ω as a function of k is provided in Canosa (1973).

We test the capability of our differentiable simulator framework by reproducing those calculations. To do so, we implement the functionality in figure 1(*c*). We choose a loss function that minimizes free energy given by

$$\mathcal{C}(\boldsymbol{x}_f) = -\sum_x \Delta x E(k, \omega)^2. \tag{D2}$$

In this test, we launch plasma waves using a ponderomotive driver in a box with periodic boundary conditions. For each optimization, the wavenumber k and the box size $x_{\max}$ change. We provide a lower bound of 0.5 and an upper bound of 1.4 to the optimization algorithm. We use the L-BFGS algorithm implemented in SciPy (SciPy 1.0 Contributors *et al.* 2020). We set $f_{\mathrm{tol}} = r_{\mathrm{tol}} = 10^{-12}$. The remaining parameter is the frequency ω that will minimize the value given by (D2) and ensure that it is within 2 decimal places of the direct solution to (D1), which is calculated using the root finder in SciPy.

We optimize for the resonant frequency for 100 random wavenumbers using gradients acquired using AD and finite difference (FD) and plot the performance. Figure 9 shows the distributions of the number of iterations required for each gradient-calculation method. We see that the finite-difference method requires significantly more evaluations where the maximum number of evaluations is larger by a factor of 2. One of the optimization runs requires 80 evaluations with FD, and 40 with AD. Similarly the mean and median values are smaller by a factor of 2 for AD in comparison with those for FD. It is important to note that these results are for single-variable searches. When the dimensionality of the search space increases, using FD becomes impractical. When using neural networks that are parametrized by $\gg \mathcal{O}(10)$ parameters, using FD is simply not possible, and one must use AD to acquire gradients.

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

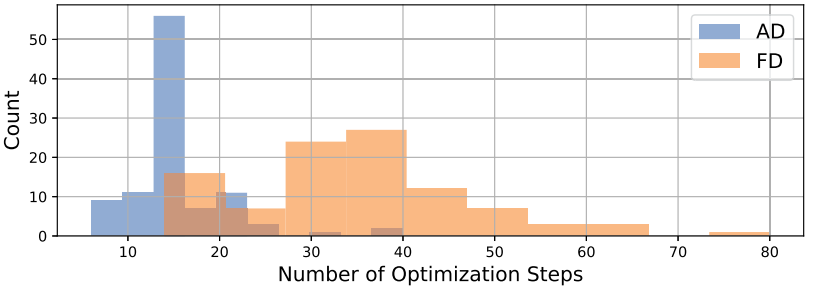


FIGURE 9. We run 100 optimizations over random wavenumbers in order to quantify the performance of gradients acquired using finite difference (FD) and AD. The plot shows a comparison of the number of iterations needed to converge to a local optimum.

REFERENCES

AFEYAN, B., CASAS, F., CROUSEILLES, N., DODHY, A., FAOU, E., MEHRENBERGER, M. & SONNENDRÜCKER, E. 2014 Simulations of kinetic electrostatic electron nonlinear (KEEN) waves with variable velocity resolution grids and high-order time-splitting. *Eur. Phys. J.* D **68** (10), 295.

ANTONSEN, T., PAUL, E.J. & LANDREMAN, M. 2019 Adjoint approach to calculating shape gradients for three-dimensional magnetic confinement equilibria. *J. Plasma Phys*. **85** (2), 905850207.

BABUSCHKIN, I., BAUMLI, K., BELL, A., BHUPATIRAJU, S., BRUCE, J. & BUCHLOVSKY, P. 2020 The DeepMind JAX Ecosystem.

BANKS, J.W., BERGER, R.L., BRUNNER, S., COHEN, B.I. & HITTINGER, J.A.F. 2011 Two-dimensional Vlasov simulation of electron plasma wave trapping, wavefront bowing, self-focusing, and sideloss. *Phys. Plasmas* **18** (5), 052102.

BAR-SINAI, Y., HOYER, S., HICKEY, J. & BRENNER, M.P. 2019 Learning data-driven discretizations for partial differential equations. *Proc. Natl Acad. Sci. USA* **116** (31), 15344–15349.

BAYDIN, A.G., PEARLMUTTER, B.A., RADUL, A.A. & SISKIND, J.M. 2018 Automatic differentiation in machine learning: a survey. *J. Machine Learning Res*. **18** (153), 1–43.

BERGER, R.L., BRUNNER, S., CHAPMAN, T., DIVOL, L., STILL, C.H. & VALEO, E.J. 2013 Electron and ion kinetic effects on non-linearly driven electron plasma and ion acoustic waves. *Phys. Plasmas* **20** (3), 032107.

BERNSTEIN, I.B., GREENE, J.M. & KRUSKAL, M.D. 1957 Exact nonlinear plasma oscillations. *Phys. Rev*. **108**, 546–550.

BHATNAGAR, P.L., GROSS, E.P. & KROOK, M. 1954 A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems. *Phys. Rev*. **94** (3), 511–525.

BRADBURY, J., FROSTIG, R., HAWKINS, P., JOHNSON, M.J., LEARY, C., MACLAURIN, D., NECULA, G., PASZKE, A., VANDERPLAS, J., WANDERMAN-MILNE, S., *et al.* 2018 JAX: composable transformations of Python+NumPy programs. http://github.com/google/jax

CANOSA, J. 1973 Numerical solution of Landau's dispersion equation. *J. Comput. Phys*. **13** (1), 158–160.

CASAS, F., CROUSEILLES, N., FAOU, E. & MEHRENBERGER, M. 2017 High-order Hamiltonian splitting for the Vlasov–Poisson equations. *Numer. Math*. **135** (3), 769–801.

CHEN, C.H.K., KLEIN, K.G. & HOWES, G.G. 2019 Evidence for electron Landau damping in space plasma turbulence. *Nat. Commun*. **10** (1), 740.

CHEN, R.T.Q., RUBANOVA, Y., BETTENCOURT, J. & DUVENAUD, D.K. 2018 Neural ordinary differential equations. In *Advances in Neural Information Processing System* (eds. S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi & R. Garnett), vol. 31. Curran Associates.

CHEN, T., XU, B., ZHANG, C. & GUESTRIN, C. 2016 Training deep nets with sublinear memory cost. arXiv:1604.06174.

CONLIN, R., DUDT, D.W., PANICI, D. & KOLEMEN, E. 2022 The DESC stellarator code suite part II: perturbation and continuation methods. arXiv:2203.15927.

DEWAR, R.L. 1972 Frequency shift due to trapped particles. *Phys. Fluids* **15** (4), 712–714.

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

DOUGHERTY, J.P. 1964 Model Fokker-Planck equation for a plasma and its solution. *Phys. Fluids* **7** (11), 1788–1799.

DUDT, D., CONLIN, R., PANICI, D. & KOLEMEN, E. 2022 The DESC stellarator code suite part III: quasi-symmetry optimization. arXiv:2204.00078.

ELLIS, I.N., STROZZI, D.J., WINJUM, B.J., TSUNG, F.S., GRISMAYER, T., MORI, W.B., FAHLEN, J.E. & WILLIAMS, E.A. 2012 Convective Raman amplification of light pulses causing kinetic inflation in inertial fusion plasmas. *Phys. Plasmas* **19** (11), 112704.

FAHLEN, J.E., WINJUM, B.J., GRISMAYER, T. & MORI, W.B. 2009 Propagation and damping of nonlinear plasma wave packets. *Phys. Rev. Lett*. **102** (24), 245002.

GRIEWANK, A. & WALTHER, A. 2000 Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Softw*. **26** (1), 19–45.

HENNIGAN, T., CAI, T., NORMAN, T. & BABUSCHKIN, I. 2020 Haiku: Sonnet for JAX. http://github.com/deepmind/dm-haiku

HE, Z.H., HOU, B., LEBAILLY, V., NEES, J.A., KRUSHELNICK, K. & THOMAS, A.G.R. 2015 Coherent control of plasma dynamics. *Nat. Commun*. **6** (1), 7156.

HOLL, P., KOLTUN, V. & THUEREY, N. 2020 Learning to control PDEs with differentiable physics. arXiv:2001.07457.

HUBA, J.D. 2011 NRL plasma formulary. *Plasma Phys*. 1–71.

JALAS, S., KIRCHEN, M., MESSNER, P., WINKLER, P., HÜBNER, L., DIRKWINKEL, J., SCHNEPP, M., LEHE, R. & MAIER, A.R. 2021 Bayesian optimization of a Laser-Plasma accelerator. *Phys. Rev. Lett*. **126** (10), 104801.

JOGLEKAR, A. & LEVY, M. 2020 VlaPy: a python package for Eulerian Vlasov-Poisson-Fokker-Planck simulations. *J. Open Source Softw*. **5** (53), 2182.

KAUFMANN, R.L. & PATERSON, W.R. 2009 Boltzmann H function and entropy in the plasma sheet. *J. Geophys. Res*. **114** (A9). https://doi-org.proxy.lib.umich.edu/10.1029/2008JA014030

KINGMA, D.P. & BA, J. 2017 Adam: a method for stochastic optimization. arXiv:1412.6980.

KOCHKOV, D., SMITH, J.A., ALIEVA, A., WANG, Q., BRENNER, M.P. & HOYER, S. 2021 Machine learning accelerated computational fluid dynamics. arXiv:2102.01010.

KRALL, J., JOYCE, G. & ESAREY, E. 1991 Vlasov simulations of very-large-amplitude-wave generation in the plasma wake-field accelerator. *Phys. Rev.* A **44**, 6854–6861.

LENARD, A. & BERNSTEIN, I.B. 1958 Plasma oscillations with diffusion in velocity space. *Phys. Rev*. **112** (5), 1456–1459.

MANHEIMER, W.M. & FLYNN, R.W. 1971 Formation of stationary large amplitude waves in plasmas. *Phys. Fluids* **14** (11), 2393–2396.

MCGREIVY, N., HUDSON, S.R. & ZHU, C. 2021 Optimized finite-build stellarator coils using automatic differentiation. *Nucl. Fusion* **61** (2), 026020.

METZ, L., FREEMAN, C.D., SCHOENHOLZ, S.S. & KACHMAN, T. 2022 Gradients are not all you need. arXiv:2111.05803.

MORALES, G.J. & O'NEIL, T.M. 1972 Nonlinear frequency shift of an electron plasma wave. *Phys. Rev. Lett*. **28** (7), 417–420.

NAUMANN, U. 2004 Optimal accumulation of Jacobian matrices by elimination methods on the dual computational graph. *Math. Program*. **99** (3), 399–421.

NOCEDAL, J. & WRIGHT, S.J. 2006 *Numerical Optimization*, 2nd edn. Springer.

OKTAY, D., MCGREIVY, N., ADUOL, J., BEATSON, A. & ADAMS, R.P. 2021 Randomized automatic differentiation. arXiv:2007.10412.

O'NEIL, T. 1965 Collisionless Damping of Nonlinear Plasma Oscillations. *Phys. Fluids* **8** (12), 2255.

PANICI, D., CONLIN, R., DUDT, D.W. & KOLEMEN, E. 2022 The DESC stellarator code suite part I: quick and accurate equilibria computations. arXiv:2203.17173.

PAUL, E.J., ANTONSEN, T., LANDREMAN, M. & COOPER, W.A. 2020 Adjoint approach to calculating shape gradients for three-dimensional magnetic confinement equilibria. Part 2. Applications. *J. Plasma Phys*. **86** (1), 905860103.

PAUL, E.J., LANDREMAN, M. & ANTONSEN, T. 2021 Gradient-based optimization of 3D MHD equilibria. *J. Plasma Phys*. **87** (2), 905870214.

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press

Pezzi, O., Cozzani, G., Califano, F., Valentini, F., Guarrasi, M., Camporeale, E., Brunetti, G., Retinó, A. & Veltri, P. 2019 ViDA: a Vlasov–Darwin solver for plasma physics at electron scales. *J. Plasma Phys.* **85** (5), 905850506.

Pontryagin, L.S. 1987 *Mathematical Theory of Optimal Processes*. CRC Press.

Salimans, T., Ho, J., Chen, X., Sidor, S. & Sutskever, I. 2017 Evolution strategies as a scalable alternative to reinforcement learning. arXiv:1703.03864.

Schoenholz, S.S., Cubuk, E.D. & Jax, M.D. 2019 End-to-end differentiable, hardware accelerated, molecular dynamics in pure python. arXiv:1912.04232.

Shalloo, R.J., Dann, S.J.D., Gruse, J. -N., Underwood, C.I.D., Antoine, A.F., Arran, C., Backhouse, M., Baird, C.D., Balcazar, M.D., Bourgeois, N., *et al.* 2020 Automation and control of laser wakefield accelerators using Bayesian optimisation. arXiv:2007.14340.

Smith, J.R., Orban, C., Morrison, J.T., George, K.M., Ngirmang, G.K., Chowdhury, E.A. & Roquemore, W.M. 2020 Optimizing laser–plasma interactions for ion acceleration using particle-in-cell simulations and evolutionary algorithms. *New J. Phys.* **22** (10), 103067.

Strozzi, D.J., Williams, E.A., Langdon, A.B. & Bers, A. 2007 Kinetic enhancement of Raman backscatter, and electron acoustic Thomson scatter. *Phys. Plasmas* **14** (1), 013104.

Thomas, A.G.R. 2016 Vlasov simulations of thermal plasma waves with relativistic phase velocity in a Lorentz boosted frame. *Phys. Rev.* E **94** (5), 053204.

SciPy 1.0 Contributors, Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., *et al.* 2020 SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat. Methods* **17** (3), 261–272.

Winjum, B.J., Tableman, A., Tsung, F.S. & Mori, W.B. 2019 Interactions of laser speckles due to kinetic stimulated Raman scattering. *Phys. Plasmas* **26** (11), 112701.

Zhang, X., Clune, J. & Stanley, K.O. 2017 On the relationship between the OpenAI evolution strategy and stochastic gradient descent. arXiv:1712.06564.

Zhuang, J., Kochkov, D., Bar-Sinai, Y., Brenner, M.P. & Hoyer, S. 2020 Learned discretizations for passive scalar advection in a 2-D turbulent flow. arXiv:2004.05477.

Zhu, C., Hudson, S.R., Song, Y. & Wan, Y. 2017 New method to design stellarator coils without the winding surface. *Nucl. Fusion* **58** (1), 016008.

https://doi.org/10.1017/S0022377822000939 Published online by Cambridge University Press