The Journal of Open Source Software

VlaPy: A Python package for Eulerian Vlasov-Poisson-Fokker-Planck Simulations

Archis S. Joglekar[1] **and Matthew C. Levy**[1]

1 Noble.AI, 8 California St, Suite 400, San Francisco, California 94111

DOI: 10.21105/joss.02182

Software

- Review
- Repository
- Archive

Editor: David P. Sanders

Reviewers:

- @TomGoffrey
- @StanczakDominik

Submitted: 29 February 2020
Published: 17 September 2020

Summary

Here we introduce `VlaPy`: a one-spatial-dimension, one-velocity-dimension (1D-1V), Eulerian Vlasov-Poisson-Fokker-Planck (VPFP) simulation code written in Python.

The Vlasov-Poisson-Fokker-Planck system of equations is commonly used to study plasma and fluid physics in a broad set of topical environments, ranging from space physics, to laboratory-created plasmas for fusion applications (Betti & Hurricane, 2016; Chen, Klein, & Howes, 2019; Fasoli et al., 2016; Ongena, Koch, Wolf, & Zohm, 2016). More specifically, the Vlasov-Poisson system of equations is typically employed to model collisionless plasmas. The Fokker-Planck operator can be introduced into this system to represent the effect of collisions. The primary advantage of this scheme is that instead of relying on numerical diffusion to smooth small-scale structures that arise when modeling collisionless plasmas, the Fokker-Planck operator enables a physics-based smoothing mechanism.

Our implementation is based on finite-difference and pseudo-spectral methods. At the lowest level, `VlaPy` evolves a two-dimensional (2D) grid according to this set of coupled partial integro-differential equations over time. In `VlaPy`, the simulation dynamics can be initialized through user-specified initial conditions or external forces.

Statement of Need

The 1D-1V VPFP equation set solved here has been applied in research on laser-plasma interactions in the context of inertial fusion (Banks, Brunner, Berger, & Tran, 2016; Fahlen, Winjum, Grismayer, & Mori, 2009), plasma-based accelerators (Thomas, 2016), space physics (Chen et al., 2019), and fundamental plasma physics (Heninger & Morrison, 2018; Pezzi, Valentini, & Veltri, 2016). While there are VPFP software libraries which are available in academic settings, research laboratories, and industry (Banks, Brunner, Berger, Arrighi, & Tran, 2017; Joglekar et al., 2018), the community has yet to benefit from a simple-to-read, open-source Python implementation. This lack of capability is currently echoed in conversations within the `PlasmaPy` (PlasmaPy Community et al., 2018) community (`PlasmaPy` is a collection of open-source plasma physics resources). Our aim with `VlaPy` is to take a step towards filling this need for a research and educational tool in the open-source community.

`VlaPy` is intended to help students learn fundamental concepts and help researchers discover novel physics and applications in plasma physics, fluid physics, computational physics, and numerical methods. It is also designed to provide a science-accessible introduction to industry and software engineering best-practices, including unit and integrated testing, and extensible and maintainable code.

The details of the `VlaPy` implementation are provided in the following sections.

Equations

The Vlasov-Poisson-Fokker-Planck system can be decomposed into four components. These components, represented using normalized units, are $\tilde{v} = v/v_{th}$, $\tilde{t} = t/\omega_p^{-1}$, $\tilde{x} = x/\lambda_D$, $\tilde{m} = m/m_e$, $\tilde{q} = q/e$, $\tilde{m} = m/m_e$, $\tilde{E} = eE/m_e v_{th}\omega_p$, $\tilde{f} = f/n_e v_{th}^{-3}$ where v_{th} is the thermal velocity, ω_p is the electron plasma frequency, m_e is the electron mass, λ_D is the Debye length, and e is the elementary charge. The Fourier transform operator is represented by $\mathcal{F}$ and the subscript to the operator indicates the dimension of the transform. In what follows, we have omitted the tilde for brevity.

Vlasov Equation

The normalized, non-relativistic ($\gamma = 1$) Vlasov equation for electrons is given by

$$\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial x} - E(x)\frac{\partial f}{\partial v} = 0,$$

where $f = f(x, v, t)$ is the electron velocity distribution function.

We use operator splitting to advance the time-step (Strang, 1968). Each one of those operators is then integrated pseudo-spectrally using the following methodology.

We use the Fourier expansions of the distribution function, which are given by

$$f(x_l, v_j) = \sum \hat{f}_x(k_x, v_j)\exp(ik_x x_l) = \sum \hat{f}_v(x_l, k_v)\exp(ik_v v_j).$$

We first discretize $f(x, v, t) = f^n(x_l, v_j)$, and then perform a Fourier expansion in $\hat{x}$ for each grid value of v.

This gives

$$f^n(x_l, v_j) = \sum \hat{f}_x^n(k_x, v_j)\exp(ik_x x_j)$$

which is substituted into the Fourier transform of the advection operator in $\hat{x}$, as given by

$$\mathcal{F}_x\left[\frac{\partial f}{\partial t} = -v\frac{\partial f}{\partial x}\right].$$

This process enables the decoupling of $\hat{x}$ and $\hat{v}$ grids from the time dimension and allows us to write an Ordinary Differential Equation in time for the discretized distribution function $\hat{f_x}^n(k_x, v_j)$. This is given by

$$\frac{d\left[\hat{f_x}^n(k_x, v_j)\right]}{\hat{f_x}^n(k_x, v_j)} = -v_j\ (ik_x)\ dt.$$

Next, we solve for the change in the plasma distribution function, integrate in time, and evaluate the integral at $\hat{f_x}^n$ and $\hat{f_x}^{n+1}$ which gives

$$\hat{f_x}^{n+1}(k_x, v_j) = \exp(-ik_x\ v_j\Delta t)\ \ \hat{f_x}^n(k_x, v_j).$$

The $E\partial f/\partial v$ term is evolved similarly using

$$\hat{f_v}^{n+1}(x_l, k_v) = \exp(-ik_v\ E_l\Delta t)\ \ \hat{f_v}^n(x_l, k_v).$$

We have implemented a simple Leapfrog scheme as well as a 4th order integrator called the Position-Extended-Forest-Ruth-Like Algorithm (PEFRL) (Omelyan, Mryglod, & Folk, 2002)

Tests

The implementation of this equation is tested in the integrated tests section.

Poisson Equation

The normalized Poisson equation is simply

$$\nabla^2 \Phi = -\rho$$

Because the ion species are effectively static and form a charge-neutralizing background to the electron dynamics, we can express the Poisson equation as

$$-\nabla E = -\rho_{net} = -(1 - \rho_e)$$

This is justifed by the assumption that the relevant time-scales are short compared to the time-scale associated to ion motion.

In one spatial dimension, this can be expressed as

$$\frac{\partial}{\partial x} E(x) = 1 - \int f(x, v)\ dv$$

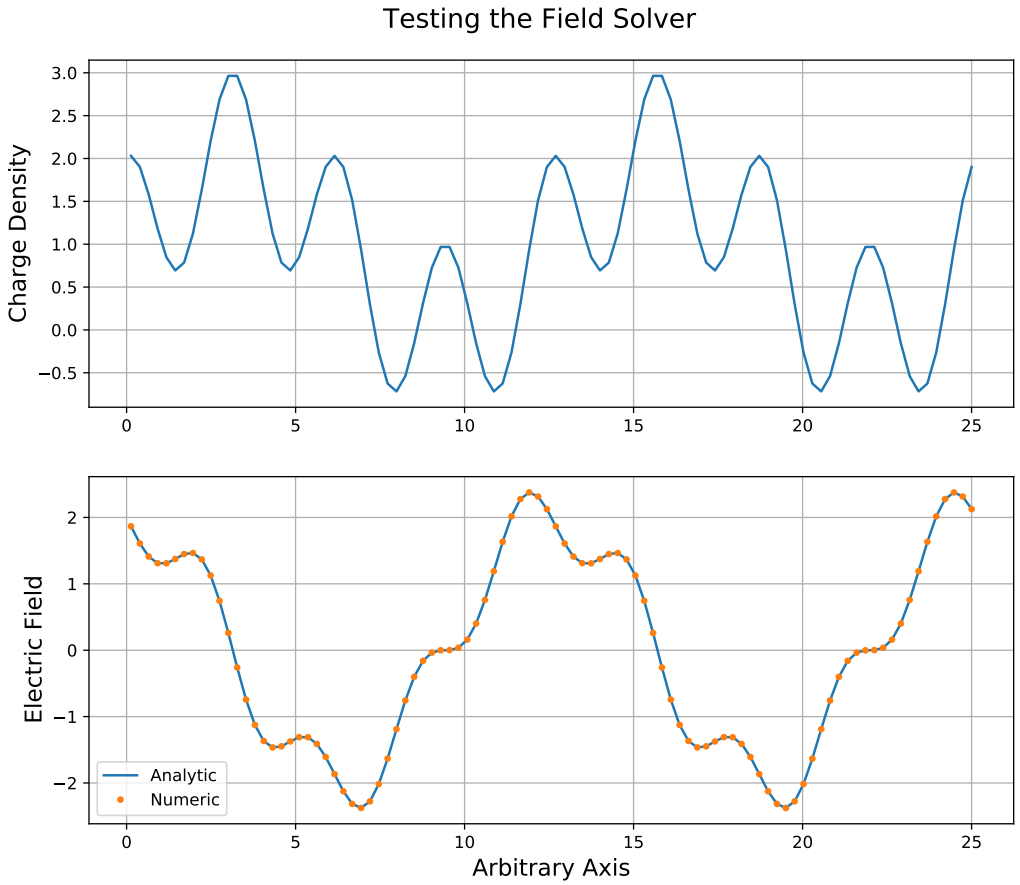
and the discretized version that is solved is

$$E(x_i)^n = \mathcal{F}_x^{-1}\left[\frac{\mathcal{F}_x\left(1 - \sum^j f^n(x_i, v_j)\Delta v\right)}{ik_x}\right]$$

Integrated Code Testing

Unit tests are provided for this operator to validate its performance and operation under the above assumptions. These are simply unit tests against analytical solutions of integrals of periodic functions. They can be found in `tests/test_fieldsolver.py`.

Below, we provide an example illustration of this validation. The code is provided in `notebooks/test_poisson.ipynb`



Fokker-Planck Equation

We have implemented two simplified versions of the full Fokker-Planck operator (Dougherty, 1964; Lenard & Bernstein, 1958).

The first of these implementations (LB) has the governing equation given by

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

where

$$v_0^2 = \int v^2 f(x,v)\ dv,$$

is the thermal velocity of the distribution.

The second of these implementations (DG) has a governing equation given by

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

where

$$\underline{v} = \int vf(x,v)\ dv,$$

is the mean velocity of the distribution and

$$v_t^2 = \int (v-\underline{v})^2 f(x,v)\ dv,$$

is the thermal velocity of the shifted distribution.

The second implementation is an extension of the first, and extends momentum conservation for distributions that have a non-zero mean velocity.

We discretize this backward-in-time, centered-in-space. This procedure results in the time-step scheme given by

$$f^n = \left[LD \times \bar{v}_{j+1} f_{j+1}^{n+1} + DI \times f_j^{n+1} + UD \times \bar{v}_{j-1} f_{j-1}^{n+1}\right].$$

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

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

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

where $\bar{v} = v$ or $\bar{v} = v - \underline{v}$ depending on the implementation.

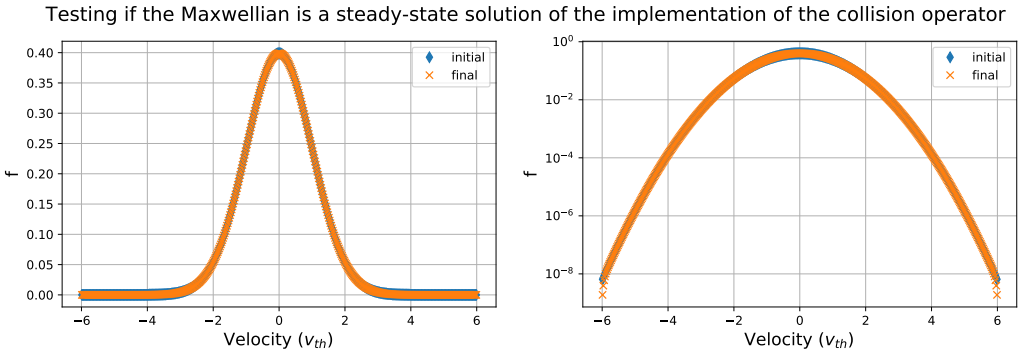
This forms a tridiagonal system of equations that can be directly inverted.

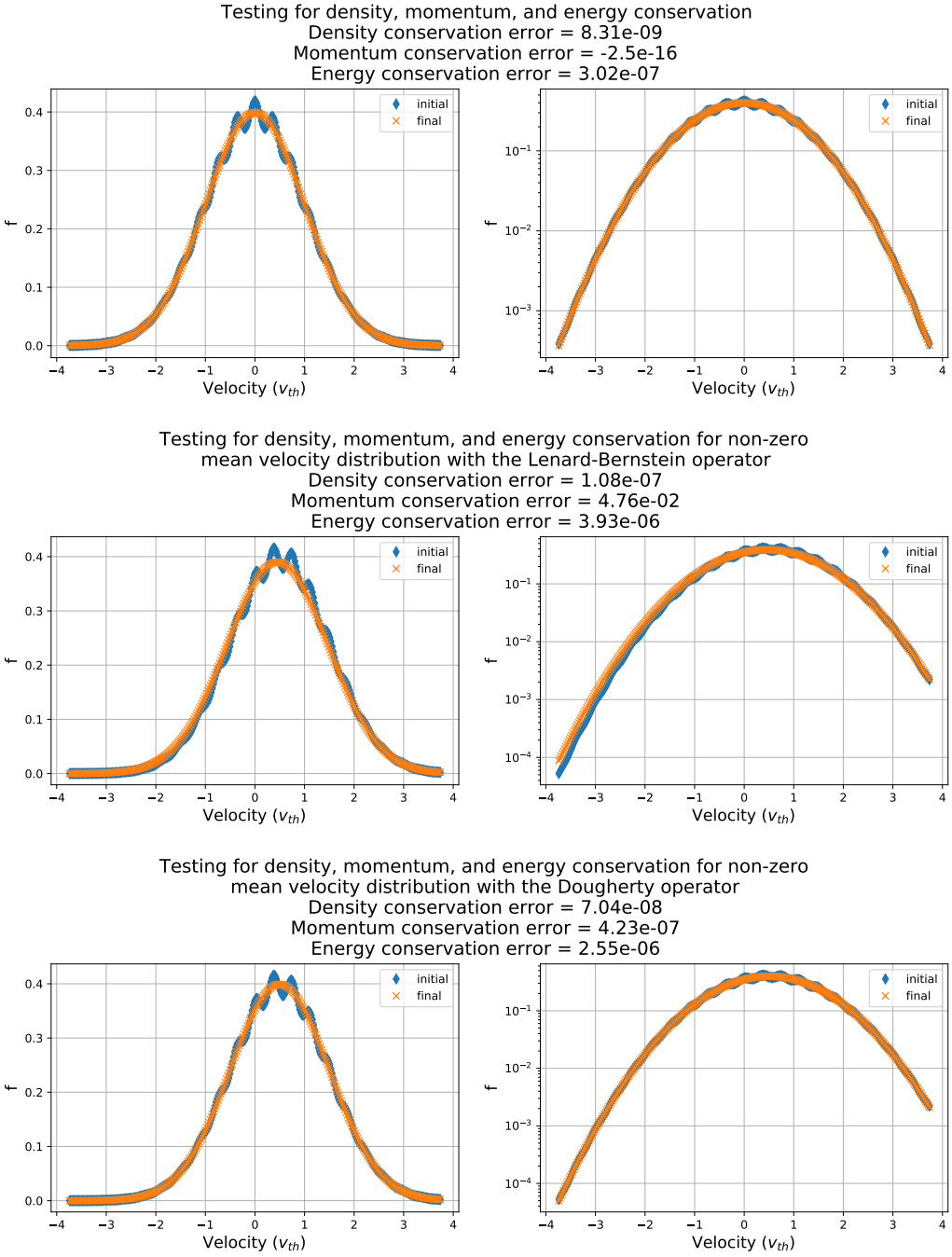
Integrated Code Testing

Unit tests are provided for this operator. They can be found in `tests/test_lb.py` and `tests/test_dg.py`. The unit tests ensure that

1. The operator does not impact a Maxwell-Boltzmann distribution already satisfying $v_{th} = v_0$.
2. The LB operator conserves number density, momentum, and energy when initialized with a zero mean velocity.
3. The DG operator conserves number density, momentum, and energy when initialized with a non-zero mean velocity.

The `notebooks/test_fokker_planck.ipynb` notebook contains illustrations and examples for these tests. Below, we show results from some of the tests for illustrative purposes.





We see from the above figures that the distribution relaxes to a Maxwellian. Depending on the implementation, certain characteristics of momentum conservation are enforced or avoided.

Integrated Code Tests against Plasma Physics: Electron Plasma Waves and Landau Damping

Landau Damping is one of the most fundamental phenomena in plasma physics. An extensive review is provided in (Ryutov, 1999).

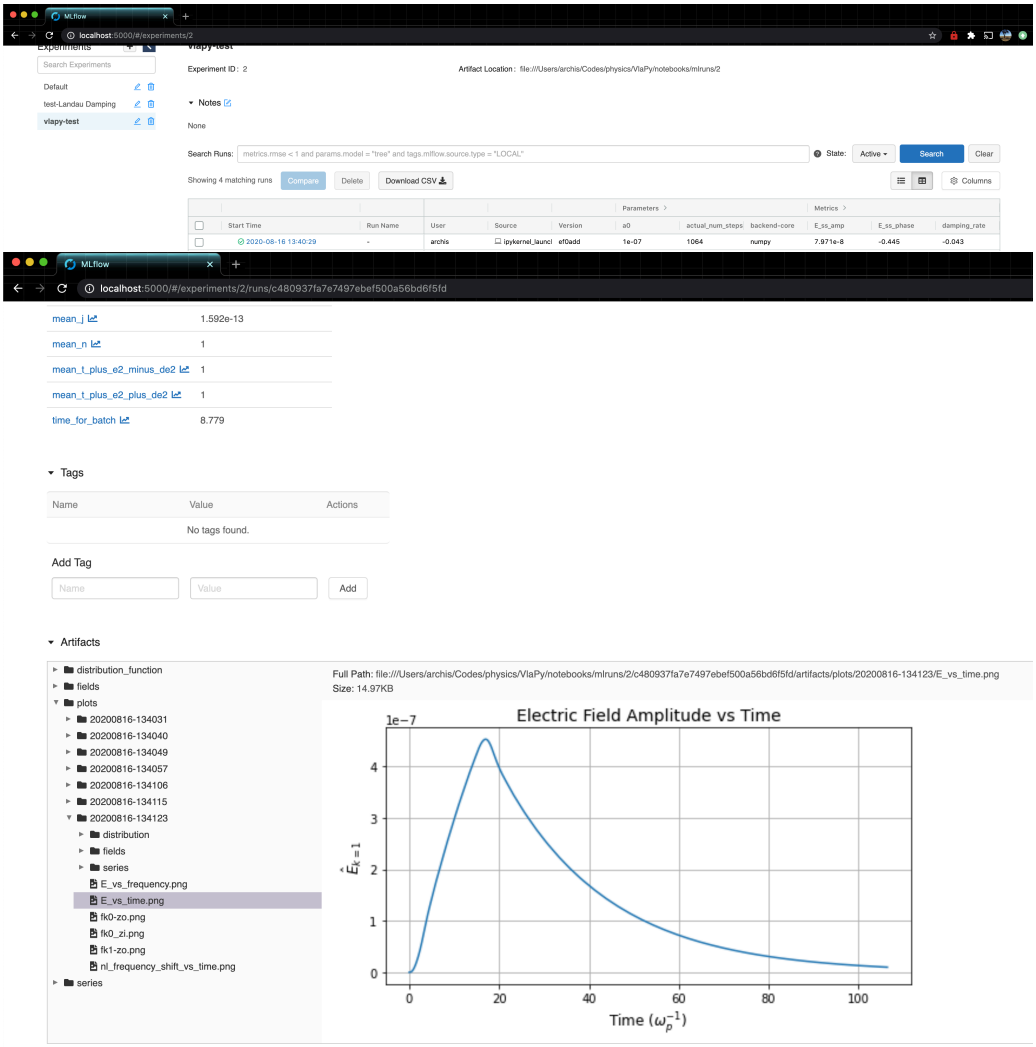
Plasmas can support electrostatic oscillations. The oscillation frequency is given by the elec-

trostatic electron plasma wave (EPW) dispersion relation. When a wave of sufficiently small amplitude is driven at the resonant wave-number and frequency pairing, there is a resonant exchange of energy between the plasma and the electric field, and the electrons can damp the electric field. The damping rates, as well as the resonant frequencies, are given in (Canosa, 1973).

In the `VlaPy` simulation code, we have verified that the known damping rates for Landau Damping are reproduced, for a few different wave-numbers. This is shown in `notebooks/landau_damping.ipynb`.

We include validation against this phenomenon as an automated integrated test. The tests can be found in `tests/test_landau_damping.py`

Below, we also illustrate a manual validation of this phenomenon through the fully integrated workflow of running a simulation on a local machine and sending the results to the MLFlow-driven logging mechanism. After running a properly initialized simulation, we show that the damping rate of an electron plasma wave with $k = 0.3$ is reproduced accurately through the UI. This can also be computed manually (please see the testing code for details).



To run the entire testing suite, make sure `pytest` is installed, and call `pytest` from the root folder for the repository. Individual files can also be run by calling `pytest tests/<test_filename>.py`.

Example Run Script For Landau Damping

```
import numpy as np
from vlapy import manager, initializers
from vlapy.infrastructure import mlflow_helpers, print_to_screen
from vlapy.diagnostics import landau_damping

if __name__ == "__main__":
    # Pick a random wavenumber
    k0 = np.random.uniform(0.3, 0.4, 1)[0]

    # This is a collisionless simulation. Provide float value if collisions should
    log_nu_over_nu_ld = None

    # This initializes the default parameters
    all_params_dict = initializers.make_default_params_dictionary()

    # This calculates the roots to the EPW dispersion relation given the wavenumbe
    all_params_dict = initializers.specify_epw_params_to_dict(
        k0=k0, all_params_dict=all_params_dict
    )

    # This specifies the collision frequency given nu_ld
    all_params_dict = initializers.specify_collisions_to_dict(
        log_nu_over_nu_ld=log_nu_over_nu_ld, all_params_dict=all_params_dict
    )

    # The solvers can be chosen here
    all_params_dict["vlasov-poisson"]["time"] = "leapfrog"
    all_params_dict["vlasov-poisson"]["edfdv"] = "exponential"
    all_params_dict["vlasov-poisson"]["vdfdx"] = "exponential"

    all_params_dict["fokker-planck"]["type"] = "lb"
    all_params_dict["fokker-planck"]["solver"] = "batched_tridiagonal"

    # The pulse shape can be chosen here
    pulse_dictionary = {
        "first pulse": {
            "start_time": 0,
            "t_L": 6,
            "t_wL": 2.5,
            "t_R": 20,
            "t_wR": 2.5,
            "w0": all_params_dict["w_epw"],
            "a0": 1e-7,
            "k0": k0,
        }
    }

    # Mlflow experiment name and location
    mlflow_exp_name = "landau-damping"

    # Either an IP address for your MLflow server or "local" if no server specifie
    uris = {
```

```
    "tracking": "local",
}


# Start!
that_run = manager.start_run(
    all_params=all_params_dict,
    pulse_dictionary=pulse_dictionary,
    diagnostics=landau_damping.LandauDamping(
        vph=all_params_dict["v_ph"],
        wepw=all_params_dict["w_epw"],
    ),
    uris=uris,
    name=mlflow_exp_name,
)

# Assess if the simulation results match the actual damping rate
print(
    mlflow_helpers.get_this_metric_of_this_run("damping_rate", that_run),
    all_params_dict["nu_ld"],
)
```

Acknowledgements

We use xarray (Hoyer & Hamman, 2017) for file storage and MLFlow (Chen et al., 2020) for experiment management. We also acknowledge the valuable work behind NumPy (Harris et al., 2020) and SciPy (Virtanen et al., 2020).

We acknowledge valuable discussions with Pierre Navarro on the implementation of the Vlasov equation.

We are grateful for the editors' and reviewers' thorough feedback that improved the software as well as manuscript.

References

Banks, J. W., Brunner, S., Berger, R. L., Arrighi, W. J., & Tran, T. M. (2017). Collisional damping rates for electron plasma waves reassessed. *Physical Review E*, *96*(4), 043208. doi:10.1103/PhysRevE.96.043208

Banks, J. W., Brunner, S., Berger, R. L., & Tran, T. M. (2016). Vlasov simulations of electron-ion collision effects on damping of electron plasma waves. *Physics of Plasmas*, *23*(3), 032108. doi:10.1063/1.4943194

Betti, R., & Hurricane, O. A. (2016). Inertial-confinement fusion with lasers. *Nature Physics*, *12*(May), 435–448. doi:10.1038/nphys3736

Canosa, J. (1973). Numerical solution of Landau's dispersion equation. *Journal of Computational Physics*, *13*(1), 158–160. doi:10.1016/0021-9991(73)90131-9

Chen, A., Chow, A., Davidson, A., DCunha, A., Ghodsi, A., Hong, S. A., Konwinski, A., et al. (2020). Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle. In *Proceedings of the fourth international workshop on data management for end-to-end machine learning*, DEEM'20. New York, NY, USA: Association for Computing Machinery. doi:10.1145/3399579.3399867

Chen, C. H. K., Klein, K. G., & Howes, G. G. (2019). Evidence for electron Landau damping in space plasma turbulence. *Nature Communications*, *10*(1). doi:10.1038/s41467-019-08435-3

Dougherty, J. P. (1964). Model Fokker-Planck Equation for a Plasma and Its Solution. *Physics of Fluids*, *7*(11), 1788. doi:10.1063/1.2746779

Fahlen, J. E., Winjum, B. J., Grismayer, T., & Mori, W. B. (2009). Propagation and damping of nonlinear plasma wave packets. *Physical Review Letters*, *102*(24), 1–4. doi:10.1103/PhysRevLett.102.245002

Fasoli, A., Brunner, S., Cooper, W. A., Graves, J. P., Ricci, P., Sauter, O., & Villard, L. (2016). Computational challenges in magnetic-confinement fusion physics. *Nature Physics*, *12*(5), 411–423. doi:10.1038/nphys3744

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., et al. (2020). Array Programming with NumPy. *Nature*, *585*(February), 357–362. doi:10.1038/s41586-020-2649-2

Heninger, J. M., & Morrison, P. J. (2018). An integral transform technique for kinetic systems with collisions. *Physics of Plasmas*, *25*(8). doi:10.1063/1.5046194

Hoyer, S., & Hamman, J. J. (2017). xarray: N-D labeled Arrays and Datasets in Python. *Journal of Open Research Software*, *5*, 1–6. doi:10.5334/jors.148

Joglekar, A. S., Winjum, B. J., Tableman, A., Wen, H., Tzoufras, M., & Mori, W. B. (2018). Validation of OSHUN against collisionless and collisional plasma physics. *Plasma Physics and Controlled Fusion*, *60*(6), 064010. doi:10.1088/1361-6587/aab978

Lenard, A., & Bernstein, I. B. (1958). Plasma oscillations with diffusion in velocity space. *Physical Review*, *112*(5), 1456–1459. doi:10.1103/PhysRev.112.1456

Omelyan, I. P., Mryglod, I. M., & Folk, R. (2002). Optimized Forest–Ruth- and Suzuki-like algorithms for integration of motion in many-body systems. *Computer Physics Communications*, *146*(2), 188–202. doi:10.1016/S0010-4655(02)00451-4

Ongena, J., Koch, R., Wolf, R., & Zohm, H. (2016). Magnetic-confinement fusion. *Nature Physics*, *12*(5), 398–410. doi:10.1038/nphys3745

Pezzi, O., Valentini, F., & Veltri, P. (2016). Collisional Relaxation of Fine Velocity Structures in Plasmas. *Physical Review Letters*, *116*(14), 1–5. doi:10.1103/PhysRevLett.116.145001

PlasmaPy Community, Murphy, N. A., Leonard, A. J., Stańczak, D., Kozlowski, P. M., Langendorf, S. J., Haggerty, C. C., et al. (2018, April). PlasmaPy: an open source community-developed Python package for plasma physics. Zenodo. doi:10.5281/zenodo.1238132

Ryutov, D. D. (1999). Landau damping: half a century with the great discovery. *Plasma Physics and Controlled Fusion*, *41*(3A), A1–A12. doi:10.1088/0741-3335/41/3A/001

Strang, G. (1968). On the Construction and Comparison of Difference Schemes. *SIAM Journal on Numerical Analysis*, *5*(3), 506–517. doi:10.1137/0705041

Thomas, A. G. R. (2016). Vlasov simulations of thermal plasma waves with relativistic phase velocity in a Lorentz boosted frame. *Physical Review E*, *94*(5), 053204. doi:10.1103/PhysRevE.94.053204

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., et al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. doi:10.1038/s41592-019-0686-2