Excalibur-Neptune report 2047356-TN-11-1

Task 3 Considerations for 1D1V models

Report covering tasks 3.1, 3.2 and 3.3

Ben Dudson, Peter Hill, Ed Higgins, David Dickinson, and Steven Wright

University of York

David Moxey

KCL

March 31, 2022

Contents

1 Executive summary

This report documents some of the work around tasks 83-3.1, 83-3.2 and 83-3.3 looking at the Vlasov-Ampere system in a periodic 1D+1V domain. A number of test cases are defined and some of the considerations around phase-space representations and time integration are discussed. Whilst simple python implemenations of this system have been produced, it is likely that future work should seek to move towards the use of existing realistic production codes, further building on the lessons learned in their development.

2 Introduction

In this work we are considering the Vlasov-Ampere system in one dimension. This is a kinetic problem and we refer to the domain as 1D1V. The goal of this work is to explore some of the considerations that should be made in designing a code to model such a system and to provide some reference implementations as a basis for future study. The document is structured as follows; suitable test cases of increasing complexity are outlined in section 3, considerations around the representation of phase-space and numerical approaches are discussed in section 4, a brief discussion of different time integration approaches is given in section 5 and we finish with a summary and look to the future in section 6. Before we progress it is helpful to set out the system and discuss normalisations.

The Vlasov-Ampere system we study (system 2-4 of reference [1]) evolves the Vlasov equation for ions and electrons and Ampere's equation for the electric field. In one dimension we may write the Vlasov equation for species s as

$$\frac{\partial f_s}{\partial t} + v_s \frac{\partial f_s}{\partial x} + \frac{q_s}{m_s} E \frac{\partial f_s}{\partial v} = 0 \tag{1}$$

and Ampere's equations as

$$\epsilon_0 \frac{\partial E}{\partial t} + J - \overline{J} = 0 \tag{2}$$

where the overline indicates a spatially average quantity and the current denisty, J, is given by

$$J = \sum_s q_s n_s u_s \tag{3}$$

with

$$n_s u_s = \int_{-\infty}^{\infty} v f_s dv \tag{4}$$

is the first fluid moment of the distribution function (i.e. bulk flow density).

Due to the short temporal and spatial scales typically associated with electron dynamics we use these as the source of our relevant normalisations. We normalise charge, mass, density and temperature to the corresponding electron values, q_e, m_e, $n_{e,0}$ and $T_{e,0}$ respectively. We normalise spatial scales to the Debye length, λ_D,

$$\lambda_D = \sqrt{\frac{\epsilon_0 T_e}{n_e q_e^2}} \tag{5}$$

and velocity scales to the electron thermal velocity[1], $v_{th,e}$

$$v_{th,e} = \sqrt{\frac{T_e}{m_e}} \tag{6}$$

. This leads to a natural temporal scale length of $1/\omega_{pe}$, with

$$\omega_{pe} = \sqrt{\frac{n_e q_e^2}{\epsilon_0 m_e}} \tag{7}$$

The electric field is normalised to E_N where

$$E_N = \sqrt{\frac{2 n_e T_e}{\epsilon_0}} \tag{8}$$

We can note that this is equivalent to normalising the energy density of the electric field, $U = \epsilon_0 E^2/2$, to the electron thermal energy density (i.e. the electron pressure, $p_e = n_e T_e$).

[1]Note there is no $\sqrt{2}$ appearing here.

This leaves the distribution function and related current density requiring normalisation. Whilst one may choose to simply normalise the distribution function to a scalar (e.g. $n_e/v_{th,e}$), it may be convenient to normalise to a velocity dependent function. Mapping $f_s \rightarrow \hat{f}_s f_N$, with f_N the distribution function normalisation and hats representing normalised quantities, we can note that

$$\frac{\partial f_s}{\partial v_s} \rightarrow \frac{1}{v_{th,e}} \frac{\partial \hat{f}_s f_N}{\partial \hat{v}_s} \tag{9}$$

Allowing for velocity dependence in f_N this becomes

$$\frac{f_N}{v_{th,e}} \left[\frac{\partial \hat{f}_s}{\partial \hat{v}_s} + \frac{\hat{f}_s}{f_N} \frac{\partial f_N}{\partial \hat{v}_s} \right] = \frac{f_N}{v_{th,e}} \left[\frac{\partial \hat{f}_s}{\partial \hat{v}_s} + \hat{f}_s \frac{\partial \log f_N}{\partial \hat{v}_s} \right] \tag{10}$$

with the latter form preferred due to avoiding the explicit division by f_N, which may become small. It should be possible to analyically precompute the constant $\frac{\partial \log f_N}{\partial v_{th,e}}$ for common choices of normalising function. We retain this form of the derivative here, noting that choosing a scalar f_N will naturally result in $\frac{\partial \log f_N}{\partial v_{th,e}} \rightarrow 0$.

The fluid flow velocity can be normalised similarly

$$n_s u_s = \hat{n}_s \hat{u}_s n_e v_{th,e} = \int_{-\infty}^{\infty} dv v f_s = \int_{-\infty}^{\infty} d\hat{v} \hat{v} \hat{f}_s f_N v_{th,e}^2 \tag{11}$$

Here we see

$$\hat{n}_s \hat{u}_s = \frac{v_{th,e}}{n_e} \int_{-\infty}^{\infty} d\hat{v} \hat{v} \hat{f}_s f_N \tag{12}$$

Introducing $\hat{f}_N = f_N v_{th,e}/n_e$ this reduces to

$$\hat{n}_s \hat{u}_s = \int_{-\infty}^{\infty} d\hat{v} \hat{v} \hat{f}_s \hat{f}_N \tag{13}$$

and we can define $\hat{J} = \sum_s \hat{q}_s \hat{n}_s \hat{u}_s$.

Under these normalisations the Vlasov equation becomes

$$\frac{\partial \hat{f}_s}{\partial \hat{t}} + \hat{v}_s \frac{\partial \hat{f}_s}{\partial \hat{x}} + \frac{\hat{q}_s}{\hat{m}_s} \hat{E} \left[\frac{\partial \hat{f}_s}{\partial \hat{v}_s} + \hat{f}_s \frac{\partial \log \hat{f}_N}{\partial \hat{v}_s} \right] = 0 \tag{14}$$

and Ampere's equation is

$$\sqrt{2}\frac{\partial \hat{E}}{\partial \hat{t}} + \hat{J} - \overline{\hat{J}} = 0 \tag{15}$$

Within this work we choose to study our normalised system, defined by equations 14, 15 and 13, in a periodic one dimensional domain with zero boundaries at $v \to \pm\infty$ and up to two plasma species. One may relax these constraints readily without changing the form of the equations.

3 Test cases

The Vlasov-Ampere system considered here evolves the distribution function for ions and electrons along with the consistent electric field. Whilst relatively simple in appearance, this system can capture a number of interesting and complex physics behaviours. Here, we outline some useful test cases for exploring the behaviour of specific implementations of this system.

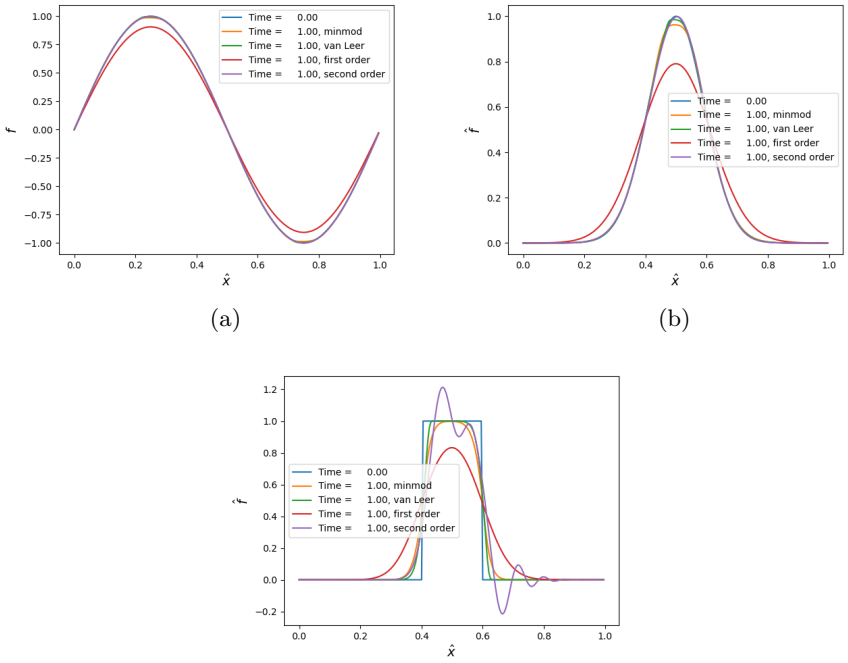
3.1 Uncoupled species : Simple 1D advection

We note that for systems in which the electric field is zero (either physically or artificially imposed), the ions and electrons are uncoupled and the individual Vlasov equations reduce to advection in one dimension. Here we propose several tests in this limit.

3.1.1 Single velocity advection

Advection of initial conditions at a single, fixed speed, v in a periodic box of length $L = 1$ for $t = NL/v$ with $N \in [1, 2, 5, 10]$ (for example). Here we expect to find the initial condition is rigidly advected back to the starting location. One can therefore measure the error by comparing the final result with the initial conditions. This should be tested for both smooth and non-smooth initial conditions. Specific examples are a sin wave, $f(x) = \sin 2\pi x/L$, a Gaussian, $f(x) = \exp -(x - L/2)^2$, and a top-hat, $f(x) = H(0.4L)[1 - H(0.6L)]$, with H

the Heaviside step function. An example for $N = 1$ is shown in figure 1 for the Finite Volume solver discussed in section 4.2. A small extension to this test is to change from periodic to Dirichlet boundary conditions. Rather than testing the function once it returns to the starting point one will need to calculate the expected location after a given time and then compare the numerical result against the initial condition shifted by the expected amount.



(a) (b)

(c)

Figure 1: Comparison of the results from a finite-volume 1D solver for periodic advection of sin (a), Gaussian (b) and top-hat (c) initial conditions for different flux limiters.

3.1.2 Multiple velocity advection

Advection of initial conditions for a grid of velocities. Each velocity is independent of all others in this system. As such, this should be equivalent to running the first test for a range of speeds. This test case enables exploration of 1D advection in the 1D1V grid setup and also introduces the impact of having a range of velocities on the time integration, for example.

3.1.3 Multiple velocity advection - phase mixing

Repeating the test of section 3.1.2 with initial conditions representing a Maxwellian multiplied by a sinusoidal variation one expects the distribution function to simply advect, remaining undamped. However, the moments of the distribution function are expected to damp out due to phase-mixing. For example, it can be shown by taking the zeroth moment of the Vlasov equation, that the density is expected to look like

$$n(t) = n(0) \exp\left[- \left(k_{\|} v_{th} t \right)^2 / 2 \right] \tag{16}$$

This phase mixing is apparent in plots of the distribution function as structures becoming increasingly sheared. This test case tests the ability to resolve these structures for a given implementation as well as allowing tests of the velocity space integration. An example of the density dependence on time from such a system simulated by GS2 [2] is shown in figure 2 for various velocity resolutions. One can see that the numerical solutions track the analytic result for increasing amounts of time as the velocity resolution is increased. Here the velocity resolution only impacts on the calculation of the density and does not enter the rest of the simulation in this setup in which the electric field has been removed[2]. The input file for the medium resolution simulation is provided in section A. The output of this test will be sensitive to numerical dissipation.

3.2 Uncoupled species : Imposed potential

The next set of test cases aim to test the advection is to explore the advection in the presence of an imposed potential or electric field. This allows one to test the coupled advection in x and v whilst avoiding complications introduced by the inclusion of a self-consistently calculated electric field. This keeps the two plasma species uncoupled such that we can study either in isolation.

[2] In other words, each velocity point is independent in the evolution of the distribution function.

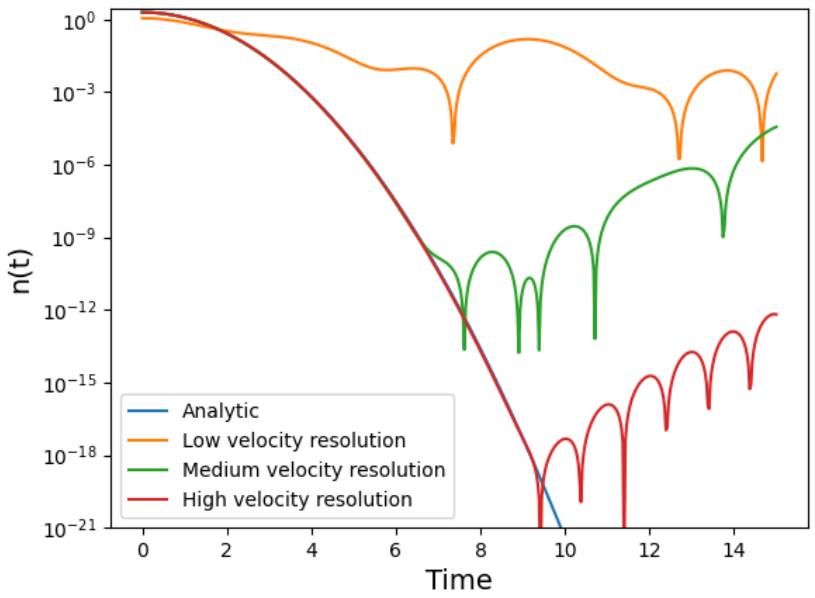


Figure 2: The density as a function of time from a periodic slab simulated by the gyrokinetic code GS2 for varying velocity resolutions, alongside the analytic solution $n(0)\exp(-t^2/2)$.

3.2.1 Uniform electric field, uniform density

The simplest test case is introduce an initial distribution function which is uniform in the parallel direction and has arbitrary structure in velocity. Introducing a uniform electric field into this system we expect to see rigid advection in the $\pm v$ direction depending on the relative sign of the field and charge. This provides a simple sanity check of the implementation, but also allows the impact of the velocity space boundary conditions to be explored.

3.2.2 Uniform electric field, varying density

A small extension to the test of section 3.2.1 is to modify the initial distribution to include smooth spatial variation. This should initially provide similar results to the earlier test described in section 3.1.2, except with shift in the velocity

which occurs at a constant rate. This should accelerate the rate at which phase mixing occurs.

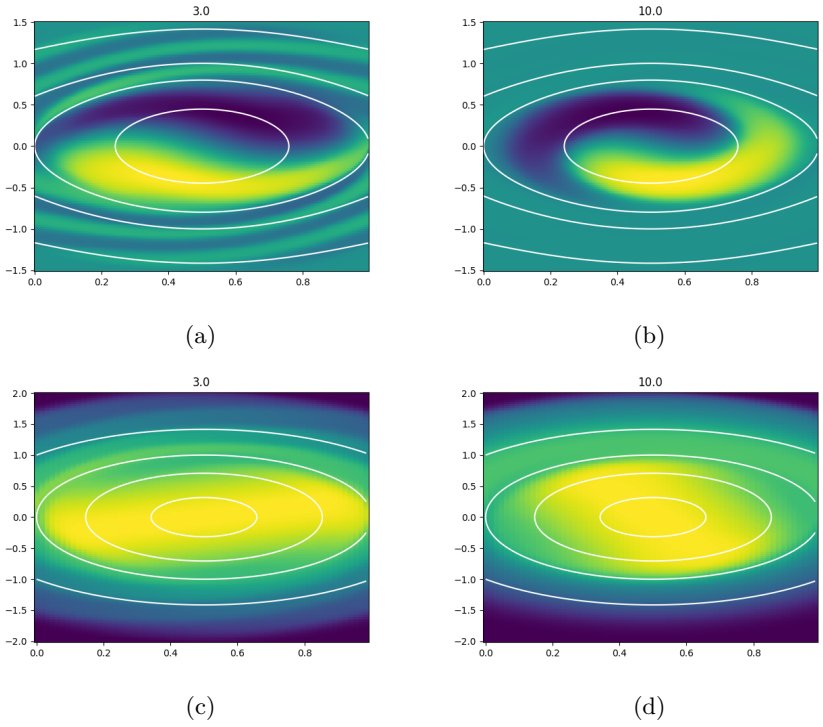
3.2.3 Spatially uniform electric field with temporal variation

The test of section 3.2.1 can be further modified to make the imposed electric field time dependent. This will cause sinusoidal motion in the velocity direction and, for appropriate settings, can take structure close to one boundary and then away from it again. This can be a good test of the numerical stability near to the boundary.

3.2.4 Spatially varying electric field

Adding a spatially varying, constant in time, electric field to the system modifies the contours of constant total energy ($\sim v^2/2 + \phi$) such that they no longer run parallel to the spatial direction. This can lead to the introduction of potential wells, seen as contours of constant total energy which are closed without crossing the parallel boundary. Particles can then become trapped in these wells and they are no longer free to sample the entire parallel space. Instead they follow closed trajectories in phase space which take them from positive velocity, to negative and back again. This is generally seen as rotating structures in phase space. The nature of this rotation depends on the structure of the imposed potential. Here we propose two main cases to test. Firstly an imposed potential of the form $\cos(x)$ which satisfies our periodic boundary conditions and is likely to play a role in realistic simulations[3]. Secondly we propose a potential of the form $(x - L/2)^2$. This is motivated to make the contours of total energy concentric circles. As a result, structures within the last closed contour (where the kinetic energy just matches the maximum potential energy) should rotate as a solid body. Examples of these two cases are given in reference [3] and sample results from a finite volume solver are shown in figure 3. One further test case would be to introduce a non-smooth electric field, such as a top-hat function. This will help test the ability to handle sharp features.

[3]This is the fundamental mode of the Fourier basis and a Fourier representation is likely to do a good job of representing all solutions in our simple periodic system.



(a) (b)

(c) (d)

Figure 3: The distribution function as a function of x (x-axis) and v (y-axis) for two different times in simulations with imposed potentials of the form $\cos 2\pi x$ (a/b) and $(x - L/2)^2$ (c/d).

3.3 Self-consistent electric field

We now turn our attention to simulations with an electric field calculated self-consistently from Ampere's equation. This case now allows electrons and ions to be coupled together and we can start to observe physical behaviour. We therefore propose simple test cases based on capturing physics phenomena.

3.3.1 Coupled species : Stationary ions - Electron plasma wave - Landau damping

We begin by considering cases in which the ions are considered stationary due to their relatively large mass. This is a good approximation for high frequency behaviour (i.e. $\omega \sim \omega_{pe}$) and short scale lengths (i.e. $L \sim \lambda_D$). Noting our

choice to normalise temporal scales to ω_{pe} and length scales to λ_D we expect stationary ions to be a good approximation whilst we consider $\mathcal{O}(1)$ normalised times and lengths. As ions are stationary they do not contribute to the current density, J, and one most simply ensure that their contribution is included in the calculation of the initial electric field to ensure self-consistent initial conditions.

One of the most basic plasma behaviour is the electron plasma oscillation. Here, a small perturbation to the electron density results in a perturbed electric field out of phase with the density perturbation. This acts to rectify the density perturbation, accelerating electrons back to the uniform equilibrium. The electrons overshoot, however, and the system reverses. This results in an oscillation with a characteristic frequency given by ω_{pe}. In a simple two-fluid picture this oscillation should continue unabated. In a kinetic system it's possible for wave-particle interactions to damp the wave through a process known as Landau damping. Taking the wavenumber of the perturbation small compared to the Debye length should minimise the damping and allow one to observe the plasma oscillation, whilst taking $k \sim \lambda_D$ will result in damping, for which there are analytic approximations[4]. This helps to test the performance of the phase-space treatment. It may be helpful to apply a parallel filter such that only one Fourier harmonic is permitted in the system.

3.3.2 Coupled species : Mobile ions - Sound wave

Finally we consider lower frequency oscillations such that ions can dominate the dynamics. In particular the plasma sound wave, in which electrons follow the ions to maintain quasineutrality, is a suitable test case. This has a similar setup to the electron plasma wave system, except the problem scale is different.

4 Phase space

Now that suitable test cases have been set out we can turn our attention to the numerical representation and solution of the system. There are many aspects which must be considered when deciding which approach to take and the resulting choice will vary depending on the motivation at hand. Here we briefly discuss some of the different competing factors before turning our attention to

the different numerical approaches.

Generally, time to solution is a primary metric motivating design decisions when developing new code. This is often a balance between the time to design, implement and test the code against the time it takes for the resulting code to run to completion. For any code which will be used by a large community it is generally worth time to carefully design and test the code, as the performance and robustness improvements gained from this will benefit many users over many simulations. The design of such a code will further be influenced by how well the problems to be solved can be constrained - in other words is there one clear system to be solved in all cases, or can the system of equations be varied somewhat. Finally, one must also consider the extensibility of the code - will the capability of the code be frozen or do the developers envisage this evolving significantly in time. Whilst these considerations may not immediately appear to directly influence the choice of phase-space treatment, these aspects can in fact be closely coupled. For example, whilst a Fourier treatment of the periodic parallel direction may appear appealing within system 2-4, building a code around this choice would prohibit a future change to non-periodic boundaries and would also have implications for the potential parallelisation of the code.

From a performance perspective, we wish for a treatment of phase-space which allows for efficient parallel scaling (e.g. neighbour limited communication) and high arithmetic intensity with loop kernels which can make efficient use of accelerators. At the same time, the approach should be memory efficient Our choice must also be able to robustly treat relevant physical behaviours, such as the appearance of shocks.

We begin by comparing broad approaches to the solution of partial differential equations (PDE); finite difference (FD), finite volume (FV) and finite element method (FEM). For all such methods one must consider how one uses an approximate solution to represent the true solution and how this approximate solution satisfies the PDE. Let us consider the simple advection problem

$$\frac{\partial f}{\partial t} + \nabla \cdot (uf) = 0 \tag{17}$$

In the FD approach we span the domain with a grid and can approximate derivatives using discrete difference of values associated on this grid. We as-

sume the solution is locally smooth, approximated by local polynomials, and that our approximate solution satisfies the PDE on the grid points but not between them. Such schemes are relatively intuitive and it is typically simple to implement compact stencils, limiting the direct coupling across the grid. Unfortunately such methods are not well suited to capturing sharp transitions, can struggle with complex boundaries and can be difficult to adapt in response to the evolution of the system. Finite volume methods move from the point based approach of FD to an element based approach, in which we split the domain into "cells" and represent the solution as the average across the cell. This allows greater flexibility in adapting the cells to the geometry of the system. In these methods the time derivative of the cell averaged value is dependent on the fluxes, uf, evaluated at the interfaces with neighbouring cells. This allows for better conservation properties than FD schemes, but adds the complication of evaluating the fluxes on the interface, where the cell averages are discontinuous. There are several possible approaches to this. One is to assume the solution takes the cell average at the centre of the cell and to then extrapolate from this point to the interface using an approximation of the derivative at this point. This provides different estimates of the flux either side of the interface and one must solve a Riemann problem to determine a unique value here[4]. Such methods are typically equivalent to FD in practice but provide a more convenient formulation. Alternatively one may seek a continuous interpolating polynomial, known as reconstruction. Consider seeking a quadratic approximation within the cell. To uniquely define a quadratic one requires three pieces of information. In a FD approach, one may take the value of the solution on the grid point of interest and either side of it. In FV rather than choosing the value of the function at three locations we choose to use the function average over three regions, specicially this cell and the cell either side. This provides a quadratic approximation valid within the cell of interest, giving a second order method for evaluating the function on the interface. The extrapolation/interpolation used in FV methods to find interface values, at second order, are prone to introducing additional extrema into the solution when the solution is not smooth. To avoid this FV methods are typically used in conjunction with flux limiters, which effectively apply further constraints to the calculation of the interface values in order to avoid new extrema and preserve positivity. Whilst it is possible to extend the FV approach to higher orders, such as through higher order reconstruction like the PPM method [5], at the expensive of coupling to more cells, this becomes increasingly challenging when going beyond one dimension.

Finite element methods seek to improve on this further. Rather than considering a cell to be a bounded region centred on a grid point, with a single value across this range, in a FEM method the cells (or elements) are bounded by grid points and we choose to represent the local solution in terms of a local basis. The ability to adapt the order of the basis and size of the elements is an appealing benefit of this classical FEM scheme. However, different elements share one grid point, coupling together all elements and resulting in a global implicit problem to be solved. There is also the complexity in choosing an appropriate basis function for the problem at hand. A final extension to this is the Discontinuous Galerkin approach to FEM (DG-FEM). Here we decouple the values on the nodes, such that each element may hold a different value here as in the FV approach. This then requires a Riemann like solve to construct a unique global solution, following similar methods as in FV. Such approaches are often referred to as spectral-hp methods, to reflect the fact that it uses high order polynomials and it is possible to adapt both the size of the elements and the order of the local basis as required. Overall DG-FEM appears to offer many of the attractive advantages that we seek, and this approach is likely to be used within future Neptune codes. For more information on different approaches and details of the DG-FEM approach see reference [6].

Whilst the DG-FEM approach is a long term goal, here we are interested in implementations that allow for the exploration of the test cases outlined in section 3 and are less concerned with the performance characteristics of the resulting implementation. We therefore discuss some of the choices to be made in FD and FV schemes for this system. Implementations of these approaches have been produced in python to allow exploration of some of these aspects. We also, however, highlight some examples of "real" community codes implementing different approaches in section 4.4 which could be used in the next stage of investigations.

4.1 Finite difference implementation

Here we consider some of the choices to be made in the construction of a FD approach to the 1D1V system. Firstly we consider the velocity space treatment.

[4] In simple problems this can be as simple as taking the value from down stream of the flow, this is a form of upwinding.

This impacts on two aspects; the advection in velocity by the electric field and the velocity space integrals used in the calculation of the flow density and the charge density for us in the calculation of the evolving and initial electric field respectively. The simplest approach is to select a uniform grid spanning the region $v \in [-v_l, v_l]$. This results in a very simple system which can make use of standard uniform finite differencing approaches. Considering the integration, an efficient choice of velocity space grid is determined by Gauss quadrature rules. In particular, our integration is over $v \in [-\infty, \infty]$ and a Gauss-Hermite quadrature is appealing. This quadrature can be used for integrals of the form

$$I = \int_{-\infty}^{\infty} \exp\left[-v^2\right] f(v) dv \tag{18}$$

We wish to evaluate

$$I = \int_{-\infty}^{\infty} v f_s(v) dv \tag{19}$$

If one evolves the full distribution function f_s then it is necessary to rescale this by $\exp v^2$ to use Gauss-Hermite quadrature. This has the potential to greatly amplify small amplitude errors near the velocity boundaries. To avoid this recall our earlier choice to map $f_s \rightarrow \hat{f}_s f_N$ with f_N a potentially velocity dependent normalisation. By choosing $f_N \propto \exp -v^2$ one sees that

$$I = \int_{-\infty}^{\infty} v f_s(v) dv = \int_{-\infty}^{\infty} \exp\left[-v^2\right] v \hat{f}_s(v) dv \tag{20}$$

and one can use a Gauss-Hermite quadrature with argument $v\hat{f}_s$. Such grids allow for highly accurate integrals but one notes that this results in a non-uniform grid spacing and removes the freedom to pack points aribtrarily. One possible approach to improve on this is to consider switching the velocity coordinate to energy and sign of parallel velocity. The energy domain can then be split into two regions. The lower, closed, region can be handled with a Legendre approach, whilst the upper region can utilites a Laguerre approach. Such an approach is used by GS2 [2] and is detailed in reference [7]. Whilst such schemes give a bit more flexibility we fundamentally give up control over our grid placement in exchange for accurate integration. This can pose problems if one wishes to pack points in a certain region. Furthermore, increasing the number of grid points in order to improve the resolution in one region impacts all other points and can cause the allowable time step to drop rapidly, due to a combination of larger maximum velocities on the grid and finer grid spacing. When considering the

test cases such as those in section 3.2 where a potential is imposed, there is no need to perform velocity integration.

Turning to the spatial direction, one can note that for a periodic system a natural approach to the periodic direction is in fact the use of a Fourier basis. Such an approach should be spectrally accurate and can be highly efficient when run in serial. This is appropriate for the small scale systems considered here, unfortunately such "global" spectral approaches do not parallelise well unless one can remain in spectral space throughout and the wavenumbers are independent (i.e. a linear problem). Instead, a finite difference method should offer improved scaling efficiency due to a local communication pattern (implemented through halo exchange). Furthermore, this is more readily adapted to a non-uniform grid which may become more useful if the periodic boundary conditions are replaced with more realistic options.

A python implementation of the normalised Vlasov-Ampere system defined by equations 14, 15 and 13, has been constructed as a part of this work. This implements uniform finite difference treatments for both directions as well as offering the option to use quadrature points in v or a Fourier treatment in x.

4.2 Finite volume implementation

As discussed previously, the FV approach offers several advantages over FD and here we discuss some of the choices made in seeking a python implementation of a FV treatment of the Vlasov-Ampere system. Firstly, one may note that whilst the system involves simultaneous advection in two directions, the advection rate in each direction is a constant. In other words the velocity is independent of position and the electric field is independent of velocity. This allows one to consider this as two separate 1D advection problems. The FV approach to 1D advection of f in x at speed u is

$$\frac{\partial \bar{f}_i}{\partial t} + \frac{1}{\delta x_i}\left[F_{x_i+1/2} - F_{x_i-1/2}\right] = 0 \tag{21}$$

where $\bar{f}_i$ is the average value stored in cell with index i, F is the flux uf and $\delta x_i = x_{i+1/2} - x_{i-1/2}$ is the size of the cell. One may then choose to either discretise the time derivative to produce a time advance scheme, or adopt the

method of lines (MoL) and use a generic ODE integrator to evolve $\bar{f}_i$. Here we prefer to take the MoL approach such that we may adopt library implementations for ODE integration, allowing us to explore different time integrators with ease. We note that some approaches to FV schemes, such as MUSCL [8], require one to discretise the time derivative directly to allow one to obtain information at the time mid-point, rather than use MoL. Whilst such approaches can offer tempting benefits, it can become harder to maintain the run time flexibility that may be desired in a code designed to tackle a wide range of problems.

All that remains, therefore, is to determine how to calculate the flux, or equivalently f, at the cell interfaces. Here we choose to use second order upwinded extrapolation with flux limiters. In practice this means that we follow the following steps

1. Assume the function takes the cell average value at the cell centre.
2. Calculate the one-sided finite difference approximation of the derivative at the cell centre, with the direction determined by the sign of the velocity. For example, if $v > 0$ we calculate $df_i/dx \sim (f_i - f_{i-1})/dx$.
3. Extrapolate from the cell centre to both interfaces using $f_{i\pm1/2} \approx f_i \pm 0.5 dx df_i/dx = f_i \pm 0.5(f_i - f_{i-1})$ (for $v > 0$). This gives us a second order estimate of the interface value for the current cell.
4. A first order estimate can be obtained as $f_{i\pm1/2} = f_i$.
5. Combine the low and high order interface values with weight determined by a flux limiter, ϕ. In other words $f_{i+1/2} = f_{i+1/2,low} + \phi(r_i)(f_{i+1/2,high} - f_{i+1/2,low})$. Here r_i is the ratio of the left and right sided derivatives.
6. Repeat for all cells.
7. The flux estimate may be different on either side of each interface. Here we choose a unique value by selecting the value downstream of the flow.

The overall approach is not overly complicated to implement but one can see that there is significantly more logic involved than in a FD approach. This can manifest as more branches within loops over cells. This can have a significant impact on performance through preventing vectorisation and costs due to branch

misprediction. Furthermore, one must choose an appropriate flux limiter. There are many available options and generally one seeks a total variation diminishing limiter in order to avoid the introduction of new extrema. Unfortunately no one limiter is the optimal choice for all systems and one will generally need to experiment for a given system. We also note that one may choose $\phi(r) = 0$ or $\phi(r) = 1$ to always select the low or high order method. Doing so one can often observe that the time integrator has to do more work when using a true limiter, rather than either fixed order approach. Figure 4 demonstrates the convergence of the FV advection solver for the test case set out in section 3.1.1 for a range of flux limiters. One can note that the limited approaches typically converge with rate in between the low (first) and high (second) order schemes for smooth advecting functions. In the case with the top hat the RMS convergence order is drastically reduced and the flux limiters before better than the fixed order schemes. Unsurprisingly, this error is dominated by the region around the sharp gradients. If instead one considers the local error away from these points one find extremely rapid convergence in the limited cases.

It is relatively straightforward to replace the extrapolation method with a reconstruction based approach whilst keep the remainder of the algorithm unchanged. This can allow for the study of higher order representations such as the PPM method [5] and we note a simple fortran implementation allowing the comparison of constant, linear and parabolic treatments is available at [9]. Further discussion of such methods for a similar system is given in reference [10], more detail on FV schemes can be found in reference [11] and [12]. Other python examples can be found as a part of the Pyro code [13], for example.

4.3 Brief comment on performance of python implementations

The use of python is commonplace when writing prototype codes due to the fast implementation cycle arising from the vast array of support packages and clear language. We note, however, the python is not typically considered especially performant. When working with numeric problems in python it is generally advised that one uses numpy arrays and avoids elementwise operations. Many numpy operations are implemented in a low level backend and as such should offer efficient behaviour. In implementing and testing the FD and FV codes intro-

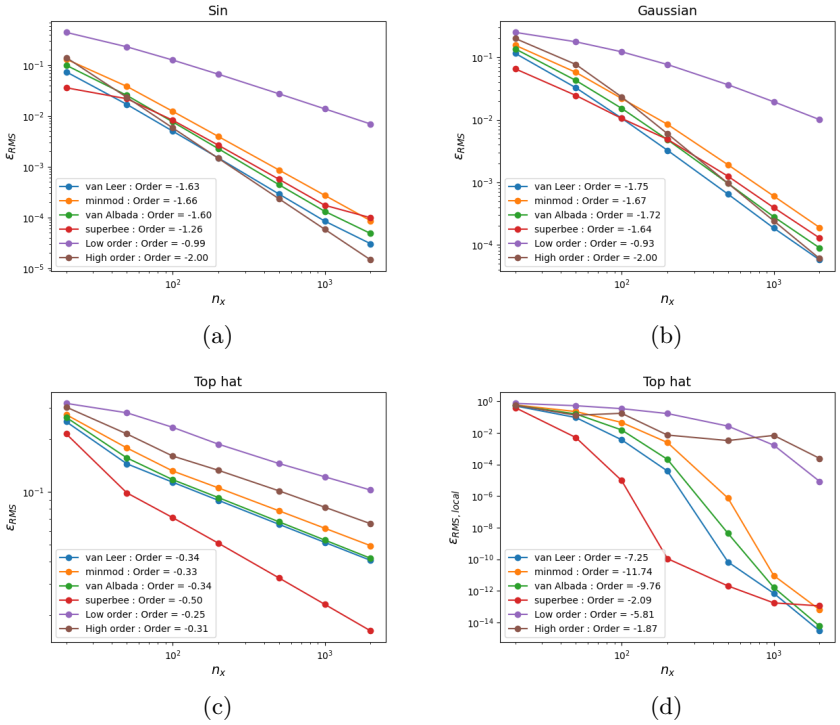


Figure 4: The RMS error as a function of grid size for the FV 1D advection solver with sin (a), Gaussian (b) and top hat (c) initial conditions. The local error at the centre of the top hat is shown in (d).

duced here it was noted that performance was rather low. Using profiling it was identified that a large fraction of the run time was spent in various simple numpy routines, such as *roll*. It numpy version 1.17 a new interface was introduced to allow external libraries to offer numpy like functionality in a transparent way [14]. Unfortunately, this appears to have introduced substantial overhead in all numpy operations. This can be seen in profiling by a prominent appearnce of the method `numpy.core._multiarray_umath.implement_array_function`. It was found that setting the environment variable `NUMPY_EXPERIMENTAL_ARRAY_FUNCTION` to 0 disables this feature and saved around 60% of run time in some test cases.

4.4 Community codes

Whilst we have discussed some of the design decisions behind the python FD and FV implementations, it is important to acknowledge the large array of existing production codes which exist in the community. Here we briefly refer to codes with which one may explore the different numerical approaches in a more realistic setting. Future work should look to incorporate some of these into the set of reference implementations.

There are many gyrokinetic finite difference codes which one may make use of. The GS2 code [2] can readily treat the 1D1V system explored here and an example for one of the test cases is shown in section 3.1.3. The stella code [15] is an actively developed flux tube gyrokinetic code. Whilst this shares some heritage with GS2, it is being used to explore different numerical approaches. In particular it uses a third order strong stability preserving RK time integrator rather than the custom implicit scheme of GS2. Furthermore, a non-interpolating semi-Lagrangian (NISL) scheme [16] is currently being developed for stella, offering the removal of smoothing associated with traditional semi-Lagrangian schemes and improving associated CFL limits. There are fewer FV production gyrokinetic codes, but the ESVM code [17] provides a useful reference implementation of the Vlasov-Maxwell system in addition to those mentioned in section 4.2. Finally it is important to highlight the Gkeyll code [18, 19]. This is a discontinuous Galerkin gyrokinetic code designed for application in the tokamak edge. This shares many of the use cases of the intended Neptune code. As such, it may be useful to use this existing implementation to further explore relevant aspects of the numerical approach. It may be useful for the Neptune team to discuss and explore collaboration with the Gkeyll team.

5 Time integration

Having constructed a discrete representation of phase-space, one needs to integrate the system in time in order to explore the dynamics. There are many choices of time integrator which could be used with each of the phase-space schemes. Within the python FD and FV implementations we make use of the *solve_ivp* method of *scipy* [20] which provides a consistent interface with which

one can employ different time integrators. For the FV scheme we found that the three different RK schemes (RK23, RK45 and DOP853) provided comparable behaviour with the lower order schemes being slightly faster but typically exhibiting larger error for the same tolerances. The flexibility to change the time integrator without large changes to the code is incredibly helpful for testing out different schemes. This is particularly important when considering codes for which different problems may be tackled. For example, in systems where a steady state is sought a low order backwards Euler method can be a good choice; the dissipation from the implicit scheme helping to damp out small rapid fluctuations and the low order resulting in a stable method with large time step. However, such an integrator is less likely to be the optimal choice for a dynamic turbulent system. With the Neptune code having a range of possible applications it is appropriate to consider how one may design the code such as to enable easy run time selection of different integrators, similar to *solve_ivp*. The PETSc library [21] provides one such interface, allowing the user full run time control over the integrator and the corresponding setup. It may be of interest however to explore novel integrators not yet available through PETSc. The BOUT++ code [22] addresses this problem with a level of abstraction such that there is a generic solver interface, for which PETSc is one of the available options. A similar approach to this may be useful in the Neptune code. We note that it is possible to utilise PETSc directly from python and it is possible to use custom integrators within the *solve_ivp* framework. Putting these together, it may be helpful to construct a wrapper to PETSc which can be used with *solve_ivp*. This would enable a study of the different time solvers available through PETSc within the simple python reference implementations.

Strong stability preserving Runge-Kutta (SSP-RK) schemes [12, 23] are a popular choice of explicit time integrator. These are constructed such as to maintain conservation properties of the underlying system. For example, in the ND generalisations of the FV schemes discussed earlier, one can guarantee conservation provided care is taken in the time discretisation used. This is often based around two first order forward-Euler half steps. This limits such approaches to first order accuracy, potentially negating some of the benefits of higher order represenations. The SSP-RK schemes are constructed from weighted combinations of forward Euler steps with weights selected to ensure conservation, good accuracy and optimal time steps. These schemes are therefore higher order, with a four stage third order scheme being particularly popular due to the relatively

large time step it affords.

The Vlasov-Ampere system contains a wide range of scales. In particular, the large ion to electron mass ratio can result in the timescales associated with electron dynamics being more than an order of magnitude shorter than the ion scales. In systems in which the electrons are required for consistency but their fast dynamics do not impact the bulk behaviour this can result in a restrictive time step when using explicit methods. More generally when there are irrelevant fast timescales implicit integrators may offer substantially larger time steps without degrading the ability to capture the relevant behaviour. Such schemes generally involve the solution of a nonlinear problem

$$M \cdot x^{n+1} = x^n \tag{22}$$

The matrix M will typically be sparse, but of course its inverse will be dense. For anything other than small problems it is often impractical to directly solve this problemand instead iterative approaches are commonly used. These are often implemented as a nested iterative structure, with each nonlinear solver iteration using a, potentially preconditioned, linear solver. These linear solvers are often based on iterative Krylov subspace methods. PETSc provides a vast array of such nonlinear and linear solvers. CVODE of SUNDIALS [24] is another commonly used nonlinear solver. Like the SSP-RK schemes, CVODE can ensure positivity is preserved through the use of constraints. There are many variants of these approaches and it can be possible to avoid storing the matrix M at all, in so called matrix-free methods. The performance of such solvers is often heavily dependent on the number of iterations required in the linear solve stage. This can be strongly influenced by the preconditioning of this system. It is possible to employ generic preconditioners, such as those provided by HYPRE [25], or physics based preconditioners (e.g. solving the bounce/transit averaged problem). Some discussion of preconditioner approaches is given in [26, 27, 28, 29]. The optimal choice is often highly problem dependent and vastly different performance can be achieved depending on how such preconditioners are set up.

Typically time integrators adapt the time step taken in order to keep the estimated error on the step below some specified tolerance. This can introduce challenges for the efficient scaling of such methods – if a global commuincation is needed at each trial time step in order to agree an error / time step then one might expect poor performance at large scale. Iterative schemes may offer

a way to side step this by choosing to iterate a sufficiently large number of times, the scheme may be able to assume that the tolerance is satisfied without communication. This may work well for well behaved systems, but not all approaches are guaranteed to converge at a fixed rate. One might anticipate that implicit schemes, able to take fewer larger time steps, will be less sensitive to this than explicit methods taking many small time steps. However, there will typically be communication involved in evaluating the right hand side of the system. One may be able to construct schemes where error estimates are propagated alongside these other communications such that the information can propagate across the communicators within a certain number of iterations. This may allow an approach where the time step is adjusted every N steps (possibly with the rejection of already attempted steps). In addition to adapting the time step it may also be helpful to adapt the order of the scheme. The domain of stability for a particular method is typically dependent on the order of the scheme[5]. An example of an adapative- step adaptive-order Adams-Bashforth solver is provided in BOUT++ [30].

An alternative approach is that of local time stepping [31]. Here different time steps can be used for different regions of the domain. This may be particularly helpful in conjuction with non-uniform/adapative phase-space approaches where a traditional time integrator may be limited everywhere by the finest resolution in the domain. Finally, we note that parallel-in-time approaches, such as ParaReal [32] and MGRIT [33], offer the appealing ability to parallelise over the temporal domain as well as the spatial domain and this may be able to make effective use of exascale resources. We note the ExCALIBUR activity in this area [34] which may provide useful input to Neptune.

6 Summary

In this report we have discussed the 1D1V system 2-4 of [1]. We begin by discussing the normalisation approach identifying the first area where the treatment of the phase-space may impact on the form of the system solved. Following this we summarise a number of increasingly challenging test cases, demonstrating sample results for some of these using both existing production codes and

[5]Multi-stage methods tend to have an increasing stable domain as the order increases, whilst multi-step methods see the opposite trend.

a simple python based finite volume code. Following this a general discussion of different numerical approaches was given introducing some of the differences between finite difference, finite volume and finite element methods. Some of the options around the treatment of phase space were discussed in the context of simple finite difference and finite volume. Examples of community codes representing these different approaches was also given and it is suggested that future work may wish to develop reference cases building on some of these existing codes, and particularly Gkeyll. We finished with a brief discussion of some of the different approaches to time integration and some of the associated challenges with going to exa-scale. Overall with the intended flexibility of the Neptune code it appears advisable to develop structures which allow easy testing of different approaches, utilising existing libraries whilst allowing novel methods to be trialled.

7 References

[1] Wayne Arter and Benjamin Dudson. Equations for ExCALIBUR/NEPTUNE Proxyapps. https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/ukaea_reports/CD-EXCALIBUR-FMS0021-1.20-M1.2.1.pdf.

[2] Michael Barnes, David Dickinson, William Dorland, Peter Alec Hill, Joseph Thomas Parker, Colin Malcolm Roach, Stephen Biggs-Fox, Nicolas Christen, Ryusuke Numata, Jason Parisi, George Wilkie, Lucian Anton, Justin Ball, Jessica Baumgaertel, Greg Colyer, Michael Hardman, Joachim Hein, Edmund Highcock, Gregory Howes, Adrian Jackson, Michael T. Kotschenreuther, Jungpyo Lee, Huw Leggate, Noah Mandell, Adwiteey Mauriya, Tomo Tatsuno, and Ferdinand Van Wyk. Gs2 v8.1.0, December 2021. Supported by CCP Plasma (https://gow.epsrc.ukri.org/NGBOViewGrant.aspx?GrantRef=EP/M022463/1) and HEC Plasma (https://gow.epsrc.ukri.org/NGBOViewGrant.aspx?GrantRef=EP/R029148/1).

[3] Ammar Hakim. A DG scheme for Vlasov equation with fixed potential. http://ammar-hakim.org/sj/je/je14/je14-vlasov-fixed-pot.html.

[4] Richard Fitzpatrick. Landau Damping. https://farside.ph.utexas.edu/teaching/plasma/lectures1/node85.html.

[5] Phillip Colella and Paul R. Woodward. The Piecewise Parabolic Method (PPM) for gas-dynamical simulations. *Journal of Computational Physics*, 54(1):174–201, 1984, doi:10.1016/0021-9991(84)90143-8.

[6] Jan S. Hesthaven and Tim Warburton. *Nodal Discontinuous Galerkin Methods*, volume 54 of *Texts in Applied Mathematics*. Springer New York, 2008.

[7] Ryusuke Numata, Gregory G. Howes, Tomoya Tatsuno, Michael Barnes, and William Dorland. Astrogk: Astrophysical gyrokinetics code. *Journal of Computational Physics*, 229(24):9347–9372, 2010, doi:https://doi.org/10.1016/j.jcp.2010.09.006.

[8] Bram van Leer. Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method. *Journal of Computational Physics*, 32(1):101–136, jul 1979, doi:10.1016/0021-9991(79)90145-1.

[9] Michael Zingale. Hydro 1D. `https://github.com/zingale/hydro1d`.

[10] T.D. Arber and R.G.L. Vann. A Critical Comparison of Eulerian-Grid-Based Vlasov Solvers. *Journal of Computational Physics*, 180(1):339–357, jul 2002, doi:10.1006/jcph.2002.7098.

[11] Michael Zingale. Introduction to Computational Astrophysical Hydrodynamics. `http://bender.astro.sunysb.edu/hydro_by_example/CompHydroTutorial.pdf`.

[12] Dale R. Durran. *Numerical Methods for Fluid Dynamics*, volume 32 of *Texts in Applied Mathematics*. Springer New York, New York, NY, 2010.

[13] Alice Harpole, Michael Zingale, Ian Hawke, and Taher Chegini. pyro : a framework for hydrodynamics explorations and prototyping. *The Journal of Open Source Software*, 4(34):1265, 2019, doi:10.21105/joss.01265.

[14] Stephan Hoyer and Matthew Rocklin and Marten van Kerkwijk and Hameer Abbasi and Eric Wieser. NEP 18 — A dispatch mechanism for NumPy's high level array functions. `https://numpy.org/neps/nep-0018-array-function-protocol.html`.

[15] Michael Barnes, Denis St-Onge, Alexander von Boetticher, Robert Davies, José Manuel García Regaña, David Dickinson, William Dorland, Peter Alec Hill, Joseph Thomas Parker, Colin Malcolm Roach, Matt Landreman, Hanne Thienpondt, Ryusuke Numata, Jason Parisi, Michael Hardman, Michael T. Kotschenreuther, and Tomo Tatsuno. stella.

[16] Harold Ritchie. Eliminating the interpolation associated with the semi-lagrangian scheme. *Monthly Weather Review*, 114(1):135 – 146, 1986, doi:10.1175/1520-0493(1986)114¡0135:ETIAWT¿2.0.CO;2.

[17] Michaël J. Touati. Esvm: an open-source finite volume electrostatic vlasov-maxwell code. *Journal of Open Source Software*, 6(67):3618, 2021, doi:10.21105/joss.03618.

[18] Gkeyll Team. Gkeyll. `https://github.com/ammarhakim/gkyl`.

[19] Ammar H. Hakim, Noah R. Mandell, T. N. Bernard, M. Francisquez, G. W. Hammett, and E. L. Shi. Continuum electromagnetic gyrokinetic simulations of turbulence in the tokamak scrape-off layer and laboratory devices. *Physics of Plasmas*, 27(4):042304, 2020, doi:10.1063/1.5141157.

[20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020, doi:10.1038/s41592-019-0686-2.

[21] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil M. Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. PETSc Web page. `https://petsc.org/`, 2021.

[22] Benjamin Daniel Dudson, Peter Alec Hill, David Dickinson, Joseph Parker, Adam Dempsey, Andrew Allen, Arka Bokshi, Brendan Shanahan, Brett Friedman, Chenhao Ma, David Bold, Dmitry Meyerson, Eric Grinaker, George Breyiannis, Hasan Muhammed, Haruki Seto, Hong Zhang, Ilon Joseph, Jarrod Leddy, Jed Brown, Jens Madsen, John Omotani, Joshua Sauppe, Kevin Savage, Licheng Wang, Luke Easy, Marta Estarellas, Matt Thomas, Maxim Umansky, Michael Løiten, Minwoo Kim, M Leconte, Nicholas Walkden, Olivier Izacard, Pengwei Xi, Peter Naylor, Fabio Riva, Sanat Tiwari, Sean Farley, Simon Myers, Tianyang Xia, Tongnyeol Rhee, Xiang Liu, Xueqiao Xu, Zhanhui Wang, Sajidah Ahmed, and Toby James. BOUT++, 3 2022.

[23] Chi-Wang Shu and Stanley Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of Computational Physics*, 77(2):439–471, 1988, doi:https://doi.org/10.1016/0021-9991(88)90177-5.

[24] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

[25] Chow, E and Cleary, A and Falgout, R. Design of the HYPRE preconditioner library. 9 1998.

[26] S Thorne. Priority equations and Test Cases. `https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2047353/TN-01.pdf`.

[27] V Alexandrov, A Lebedev, E Sahin, S Thorne. Linear systems of equations and preconditioners relating to the NEPTUNE Programme. `https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2047353/TN-02.pdf`.

[28] M Abalenkovs, V Alexandrov, A Lebedve, E Sahin, S Thorne. Implicit factorisation preconditioners for NEPTUNE programme. `https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2047353/TN-03.pdf`.

[29] M Abalenkovs, V Alexandrov, A Lebedve, E Sahin, S Thorne. Implicit factorisation preconditioners for non-symmetric problems. `https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2047353/TN-04.pdf`.

[30] David Dickinson. Adaptive Adams-Bashforth solver in BOUT++. `https://github.com/boutproject/BOUT-dev/tree/next/src/solver/impls/adams_bashforth`.

[31] Siegfried Müller and Youssef Stiriba. Fully adaptive multiscale schemes for conservation laws employing locally varying time stepping. *J. Sci. Comput.*, 30(3):493–531, mar 2007, doi:10.1007/s10915-006-9102-z.

[32] Jacques-Louis Lions, Yvon Maday, and Gabriel Turinici. Resolution d'edp par un schema en temps pararéel. *Comptes Rendus de l'Academie des Sciences - Series I - Mathematics*, 332(7):661–668, 2001, doi:https://doi.org/10.1016/S0764-4442(00)01793-6.

[33] S Friedhoff, R D Falgout, T V Kolev, S MacLachlan, and J B Schroder. A multigrid-in-time algorithm for solving evolution equations in parallel. 12 2012.

[34] Exposing parallelism: Parallel in Time. https://excalibur.ac.uk/projects/exposing-parallelism-parallel-in-time/.

A Example GS2 input file : Periodic slab test case

```
&collisions_knobs
collision_model = 'none'
/

&dist_fn_knobs
  boundary_option = 'self-periodic'
  opt_source = T
  /

  &dist_fn_species_knobs_1
  bakdif                                  =  0.00000000000E+00
  fexpr                                   =  0.50000000000E+00
  /

  &gs2_diagnostics_knobs
  nwrite = 10
  omegatol = -1.0
  write_moments = T
  write_ascii = F
  /

  &init_g_knobs
  ginit_option = 'single_parallel_mode'
  ikpar_init = 1
  chop_side = F
  phiinit = 1.0
  /

  &knobs
  delt = 0.003
  nstep = 5000
  fphi = 1.0
  /
```

```
  &kt_grids_knobs
  grid_option = 'single'
  /
  &kt_grids_single_parameters
  aky                                     =  1.0e-3
  theta0                                  =  0.00000000000E+00
  /
  &le_grids_knobs
  nesub                                   =  12
  nesuper                                 =  4
  npassing                                =  16
  /

  &source_knobs
  source_option                           =  "homogeneous"
  /
  &species_knobs
  nspec                                   =  1
  /
&species_parameters_1
  bess_fac                                =  0.10000000000!E-03
  fprim                                   =  0.0
  mass                                    =  0.10000000000E+01
  tprim                                   =  0.0
  type                                    =  "ion"
  z                                       =  0.10000000000E+01
/
&theta_grid_knobs
  equilibrium_option                      =  "s-alpha"
/

&theta_grid_parameters
  eps                                     =  0.0
  epsl                                    =  0.0
  kp                                      =  1.0
  nperiod                                 =  1
  ntheta                                  =  32
```

```
  qinp                                    =  0.0
  shat                                    =  0.0
/
&theta_grid_salpha_knobs
  model_option                            =  "no-curvature"
/
```