# Implementation and Validation of the Magnetized Poisson-Vlasov-Fokker-Planck Framework

Fabiano Sasselli

Master of Science in Computational Science and Engineering

Semester Paper HS 22

Institute of Fluid Dynamics
ETH Zürich

Supervisor: Kyoungseoun Chung

Professor: Prof. Dr. Patrick Jenny

## Abstract

In the context of the Semester Paper, the stochastic plasma simulation Python framework developed by researchers at the Institute of Fluid Dynamics (IFD), has been debugged, tested and extended to enable the time accurate and conservative simulation of plasma with particle collisions; modeled by the magnetized Poisson Vlasov-Fokker-Planck equation.

This report aims to introduce the reader to the computational method used and document the implemented framework. In the first part the reader is reminded of the common methods and models used for plasma simulations. More specifically, the model developed by IFD researchers is explained. In the second chapter specific details regarding the simulation framework are discussed. Finally in the third and fourth chapter the test results are shown and conclusions made respectively.

# Contents

# 1 Numerical simulation of the magnetized Vlasov-Fokker-Planck (VFP) Equation

## 1.1 Introduction

In presence of fluid flows with large Knudsen number ($Kn \gg 1$) (rarefied gas, small devices) the continuum assumptions made in the classical Navier-Stokes equation fails; temperature jumps, velocity slips and shocks become more prominent and the fluid is not in thermodynamic equilibrium anymore [10]. Simulating plasma not only requires tackling the difficulties listed above, but it also requires to resolve a wide range of time and length scales arising from the presence of multiple species, electromagnetic interactions between particle and external electromagnetic field and particles Coulomb collisions.

Such system can only be simulated correctly using high degree of freedom models based on statistical mechanics, i.e kinetic models based on the Boltzmann transport equation (BTE) describing the space and time evolution of the Mass Density Function (MDF) for species $s$ $\mathcal{F}_s(\mathbf{U}, \mathbf{x}, t) = \rho_s(\mathbf{x}, t) f_s(\mathbf{U}; \mathbf{x}, t)$ [3]:

$$\frac{\partial \mathcal{F}_s}{\partial t} + U_i \frac{\partial \mathcal{F}_s}{\partial x_i} + \frac{\partial \mathcal{F}_s F_i}{\partial U_i} = \left(\frac{\partial \mathcal{F}_s}{\partial t}\right)_{coll} = \sigma^{coll} \tag{1.1}$$

$U_i$ : velocity vector $i$-th coordinate
$x_i$ : position vector $i$-thcoordinate
$F_i$ : force vector $i$-th coordinate

with Boltzmann collision integral defined as [2]

$$\sigma^{coll} = \frac{1}{m_s} \int_{\mathbb{R}^3} \int_0^{4\pi} (\mathcal{F}_s(\mathbf{U}')\mathcal{F}_s(\mathbf{U}_t') - \mathcal{F}_s(\mathbf{U})\mathcal{F}_s(\mathbf{U}_t)) g I(\Omega, g) d\Omega dU_{t,1} dU_{t,2} dU_{t,3} \tag{1.2}$$

$\Omega$ : collision angle
$I(\Omega, g)$ : differential cross section
$\Omega$ : solid angle about $\mathbf{g}$
$\mathbf{g} = |\mathbf{U}' - \mathbf{U}_t'|$
$\mathbf{U}_t'$ : post-collision velocity vector

This definition of collision operator provides the most comprehensive description of Coulomb interactions between particles; but due to the high dimensionality of the integral, its direct evaluation is computationally expensive. Efforts have been made therefore in order to simplify the collision operator. *Rosenbluth et al* (1957) [5] derived a simplified operator based on the

Fokker-Planck equation:

$$\sigma^{coll,RFP} = -\Gamma_s \frac{\partial}{\partial U_i} \underbrace{\left( \mathcal{F}_s \frac{\partial h_s}{\partial U_i} \right)}_{\text{drift}} + \frac{1}{2} \frac{\partial^2}{\partial U_i \partial U_j} \underbrace{\left( \mathcal{F}_s \frac{\partial^2 g_s}{\partial U_i \partial U_j} \right)}_{\text{diffusion}} \tag{1.3}$$

$\Gamma_s$ : model coefficient
$h_s, g_s$ : model functions

This operator is is able to capture the full non-linear behaviour of collisions but requires to solve a costly four dimensional PDE with derivatives up to the fourth order introduced by the drift and diffusion terms. Further simplifications can be made by linearizing the drift term and taking a constant diffusion, leading to the linear drift-constant diffusion Fokker-Planck collision operator, which is the basis of the considered framework [3]:

$$\sigma^{coll,FP} = \frac{\partial}{\partial U_i} \left( \frac{1}{\tau} (U_i - \langle \mathbf{U}_s \rangle_i) \mathcal{F}_s \right) + \frac{\partial^2}{\partial U_k \partial U_k} \left( \frac{2 e_{s,s}}{3\tau} \mathcal{F}_s \right) \tag{1.4}$$

$\tau$ : relaxation time
$\langle \mathbf{U}_s \rangle$ : mean bath velocity (gas velocity)
$e_{s,s}$ : sensible energy for species $s$

## 1.2 Magnetized Vlasov-Fokker-Planck equation

Electromagnetic interactions can be considered by introducing in the BTE force term the Lorentz force

$$\mathbf{F} = q_s (\mathbf{E} + \mathbf{U}_s \times \mathbf{B}) \tag{1.5}$$

$q_s$ : electric charge of species $s$
$\mathbf{E}$ : electric field
$\mathbf{B}$ : magnetic field

The BTE with Lorentz forces and no collisions is commonly known as the Vlasov equation, and it will be referenced to in future sections. Coulomb collisions are included through the use of the linear drift-constant diffusion Fokker-Planck operator. Putting everything toghether and considering a time invariant external magnetic field leads to the magnetized Vlasov-Fokker-Planck (VFP) equation for species $s$ [1]:

$$\frac{\partial \mathcal{F}_s}{\partial t} + U_i \frac{\partial \mathcal{F}_s}{\partial x_i} + \frac{\partial}{\partial U_i} ((a_{s,i} - b_{s,ik} U_k) \mathcal{F}_s) = \frac{\partial^2}{\partial U_i \partial U_j} \left( \frac{c_{s,ik} c_{s,jk}}{2} \mathcal{F}_s \right) \tag{1.6}$$

with force $(\mathbf{a}_s, \mathbf{b}_s, \mathbf{c}_s)$ coefficients defined by:

$$\mathbf{a}_s = \frac{q_s}{m_s} \mathbf{E} + \eta_s \langle \mathbf{U}_s \rangle \quad ; \quad \mathbf{b}_s = -\frac{q_s}{m_s} \tilde{\mathbf{B}} + \eta_s \mathbf{I} \quad ; \quad \mathbf{c}_s = \left( 2\eta_s \frac{k_b T}{m_s} \right)^{1/2} \mathbf{I}$$

$$\tilde{\mathbf{B}} = \begin{bmatrix} 0 & -B_3 & -B_2 \\ -B_3 & 0 & B_1 \\ B_2 & -B_1 & 0 \end{bmatrix}$$

$m_s$ : mass of species $s$ elementary particle
$\eta_s$ : friction coefficient for species $s$
$k_b$ : Boltzmann constant ($1.38064852 \cdot 10^{-23}\ m^2 kg/(s^2 K)$)
$T$ : temperature

Directly solving eqn.(1.6) using classic discretization method (FDM, FVM) is expensive and prone to numerical diffusion. The impractical computational cost is caused by the required discretization of the highly dimensional phase space (7 dimensions, 3 for position, 3 for velocity and one for time). The general approach to simulate such equations consist of solving an evolution equation describing the evolution of position and velocity of computational particles inside our domain. Each computational particle represent a cloud of physical particles and can interact (dynamic collisions) with other computational particles and boundaries. By evolving in time and space the position and velocity of the computational particles, we can obtain a statistically equivalent description of the evolution of the MDF [6]:

$$\mathcal{F}_s(\mathbf{x}_s, \mathbf{U}_s, t) \approx \sum_{p=1}^{N_p} w\delta(\mathbf{x}_s - \mathbf{x}_p(t))\delta(\mathbf{U}_s - \mathbf{U}_p(t)) \quad ; \quad w = \frac{N_{p,real}}{N_p}$$

$w$ : statistical weight
$N_{p,real}$ : number of physical (real) particles
$N_p$ : number of computational particles
$\mathbf{x}_p, \mathbf{U}_p$ : position and velocity and vectors for particle $p$ of species $s$

The stochastic Langevin formulation for the evolution of the computational particles can be derived directly from the considered VFP equation [1]:

$$d\mathbf{x}_p = \mathbf{U}_p dt \tag{1.7}$$

$$d\mathbf{U}_p = (\mathbf{a}_p - \mathbf{b}_p \mathbf{U}_p)dt + \mathbf{c}_p d\mathbf{W}_p \tag{1.8}$$

$(\mathbf{a}_p, \mathbf{b}_p, \mathbf{c}_p)$ : coefficients $(\mathbf{a}_s, \mathbf{b}_s, \mathbf{c}_s)$ for particle $p$ of species $s$
$d\mathbf{W}_s$ : vector of Wiener processes increments with $\langle dW_{p,i} \rangle = 0$, $\langle dW_{p,i} dW_{p,j} \rangle = dt\delta_{ij}$

The challenge arising when trying to solve numerically the velocity process, is the intra-dimensional coupling caused by the vector product between velocity and magnetic field. For example, to find e.g $U_{s,1}$ we need to solve an equation depending on $U_{s,2}, U_{s,3}$, which are also unknown. In the next section the numerical method proposed by *Jenny and Gorji (2019)* to solve this problem is presented.

## 1.3 Accurate time integration scheme

In this section the accurate time integration scheme for the VFP eqn. developed by *Jenny and Gorji (2019)* [1] is summarized. The Langevin equations (1.7) and (1.8) are decoupled through

diagonalization of the $\mathbf{b}_s$ matrix:

$$\mathbf{b}_p = \mathbf{R}_p \boldsymbol{\beta}_p \mathbf{R}_p^{-1} \quad ; \quad \boldsymbol{\beta}_p = diag(\beta_{p,1}, \beta_{p,3}, \beta_{p,3}) \tag{1.9}$$

with $\beta_{s,\bullet}$ being an eigenvalue of $\mathbf{b}_s$and $\mathbf{R}_s$ the eigenvector matrix. The decoupled equations for particle $p$ of species $s$ can now be rewritten as:

$$dR_p^{-1} x_p = \mathbf{R}_p^{-1} \mathbf{U}_p dt \tag{1.10}$$

$$d\mathbf{R}_p^{-1}\mathbf{U}_p = (\mathbf{R}_p^{-1}\mathbf{a}_p - \boldsymbol{\beta}_p \mathbf{R}_p^{-1}\mathbf{U}_p)dt + \mathbf{R}_p^{-1}\mathbf{c}_p d\mathbf{W}_p \tag{1.11}$$

using the substitution $\hat{\mathbf{x}}_p = \mathbf{R}_p^{-1}\mathbf{x}_p$, $\hat{\mathbf{U}}_p = \mathbf{R}_p^{-1}\mathbf{U}_p$, $\boldsymbol{\alpha}_p = \boldsymbol{\beta}_p^{-1}\mathbf{R}_p^{-1}\mathbf{a}_p$, $\boldsymbol{\gamma}_p = \mathbf{R}_p^{-1}\mathbf{c}_p$ we get the transformed system:

$$d\hat{\mathbf{x}}_p = \hat{\mathbf{U}}_p dt \tag{1.12}$$

$$d\hat{\mathbf{U}}_p = \boldsymbol{\beta}_p(\boldsymbol{\alpha}_p - \hat{\mathbf{U}}_p)dt + \boldsymbol{\gamma}_p d\mathbf{W}_p \tag{1.13}$$

The authors now make the assumptions that $\boldsymbol{\alpha}_p, \boldsymbol{\beta}_p, \boldsymbol{\gamma}_p$ are frozen coefficients; i.e they do not change during a timestep. With this assumption and other manipulations, we get the decoupled exactly integrated equations for timestep $t^{n+1}$, $\Delta t = t^{n+1} - t^n$:

$$\hat{\mathbf{x}}_p^{n+1} = \hat{\mathbf{x}}_p^n + \mathbf{f} + \mathbf{B}^{1/2}\boldsymbol{\xi}_{p,x}^{n+1} \quad ; \quad \hat{\mathbf{x}}_p^{n+1} = \hat{\mathbf{x}}_p(t^{n+1}) \tag{1.14}$$

$$\hat{\mathbf{U}}_p^{n+1} = \mathbf{d} + \mathbf{A}^{1/2}\boldsymbol{\xi}_{p,u}^{n+1} \quad ; \quad \hat{\mathbf{U}}_p^{n+1} = \hat{\mathbf{U}}_p(t^{n+1}) \tag{1.15}$$

with coefficients expressed component-wise $\forall\ i, j = 1, 2, 3$

$$d_i = \alpha_i + (\hat{U}_{p,i}^n - \alpha_i)\exp[-\beta_{(i)}\Delta t]$$

$$A_{ij} = \gamma_{ik}\gamma_{jk}\frac{1}{\beta_{(i)} + \beta_{(j)}}\left(1 - \exp[-(\beta_{(i)} + \beta_{(j)})\Delta t]\right)$$

$$f_i = \alpha_i \Delta t + (\hat{U}_{p,i}^n - \alpha_i)\frac{1}{\beta_{(i)}}(1 - \exp[-\beta_{(i)}\Delta t])$$

$$B_{ij} = \frac{\gamma_{ik}\gamma_{jk}}{\beta_{(i)}\beta_{(j)}}\left(\Delta t + \frac{\exp[-\beta_{(i)}\Delta t] - 1}{\beta_{(i)}} + \frac{\exp[-\beta_{(j)}\Delta t] - 1}{\beta_{(j)}} - \frac{\exp[-(\beta_{(i)} + \beta_{(j)})\Delta t] - 1}{\beta_{(i)} + \beta_{(j)}}\right)$$

with $\beta_{(i)}$ indicating the $i$-th eigenvector of matrix $\boldsymbol{\beta}_p$. The random vector are defined as follows:

$$\boldsymbol{\xi}_{p,x}^{n+1} = \boldsymbol{\xi}_{p,1}^{n+1}$$

$$\boldsymbol{\xi}_{p,u}^{n+1} = \mathbf{F}\boldsymbol{\xi}_{p,1}^{n+1} + \mathbf{F}\boldsymbol{\xi}_{p,2}^{n+1}$$

with coefficients:

$$\mathbf{F} = \mathbf{A}^{-1/2}\mathbf{C}\mathbf{B}^{-1/2} \quad ; \quad \mathbf{G}\mathbf{G}^T = \mathbf{I} - \mathbf{A}^{-1/2}\mathbf{C}^T\mathbf{B}^{-1}\mathbf{C}\mathbf{A}^{-1/2}$$

$$C_{ij} = \frac{\gamma_{ik}\gamma_{jk}}{\beta_{(j)}}\left(\frac{1 - \exp[-\beta_{(j)}\Delta t]}{\beta_{(j)}} + \frac{\exp[-(\beta_{(i)} + \beta_{(j)})\Delta t] - 1}{\beta_{(i)} + \beta_{(j)}}\right)$$

The specific derivation of the coefficients $\mathbf{d} = \mathbf{d}(\hat{\mathbf{U}}_p^n), \mathbf{A}, \mathbf{B}, \mathbf{f} = \mathbf{f}(\hat{\mathbf{U}}_p^n)$ and random vectors $\boldsymbol{\xi}_{p,\bullet}^{n+1}$ will not be discussed here and can be found in *Jenny and Gorji (2019)* [1]. The interesting observation regarding these parameters is that they have been computed starting from conditional expectations, conditional covariance and correlation between position and velocity at different timesteps; this ultimately allows to obtain correlated and statistical consistent particle position and velocity evolution.

The proposed scheme is statistically exact only when the frozen macroscopic quantity assumption hold, i.e there is no variation of macroscopic quantities along particles paths within a timestep. Since the coefficients are thus the same for particles of the same species in the same cell; the computation happens only once per timestep and does not scale with the number of particles, leading to an increased computational efficiency. Unfortunately the frozen macroscopic quantity assumption often fails due to spatially inhomogeneous electromagnetic field [1]. In addition, the imperfect sampling of the pseudo-random vectors present in the position and velocity processes leads to a slightly biased (shifted mean) velocity distribution. The scheme takes into account collisions but not in a binary manner. Since collisions between particles are assumed to be elastic, the collision symmetries have to be fulfilled, i.e. global sum of momentum and energy transferred in between particles should be conserved. This conservation requirements have to be enforced separately through a velocity correction procedure.

## 1.4 Energy and Momentum preserving scheme

IFD researchers have proposed a new approach aimed at correcting particles velocities so that momentum and energy are conserved. The objective is to first compute a scalar correction field $\tilde{R}(\mathbf{x})$ for each quantity of interest and then compute a velocity correction coefficient $\kappa$ for each mesh cell. The correction field evaluated at particles positions has to return the corrections required to ensure that global conservation is achieved. The end goal of the researchers is to apply the correction field to the exact time integrated velocity equation shown in eqn. (1.4), (1.5).

### 1.4.1 Momentum correction

For each $p-$th particle out of all $N_p$ particles, and for each velocity component, the collision velocity is computed as the difference between the velocity predicted by the VFP model and the collisionless Vlasov model:

$$R_p = Q_p^{VFP} - Q_p^V \quad ; \quad Q_p^\bullet \in \{U_{p,1}^\bullet, U_{p,2}^\bullet, U_{p,3}^\bullet\} \tag{1.16}$$

An unknown scalar correction field is subtracted to the collision velocity $R_p$ in order to obtain a corrected collision velocity $R_p^{corr}$

$$R_p^{corr} = R_p - \tilde{R}(\mathbf{x}_p) \tag{1.17}$$

Collisions between particles are assumed to be happening in a binary manner, i.e momentum only transferred between two particles at the time. This assumption allows us to derive a simplified

momentum conservation criteria for the corrected collision velocity:

$$\sum_{p=1}^{N_p} R_p^{corr} \stackrel{!}{=} 0 \tag{1.18}$$

A continuous test function $v(\mathbf{x}) \in C^0(\Omega)$ is multiplied to the above relation. Of interest are the correction values at particles positions, therefore the test function is also evaluated at those locations:

$$\sum_{p=1}^{N_p} [R_p - \tilde{R}(\mathbf{x}_p)] v(\mathbf{x}_p) \stackrel{!}{=} 0 \tag{1.19}$$

We define a basis $B_h = \{\phi_1(\mathbf{x}), ..., \phi_N(\mathbf{x})\}$, $N = dim(V_h)$, $span\{B_h\} = V_h$, of our finite dimensional vector space $V_h$ and express both the unknown correction field and test function with respect to it:

$$\tilde{R}(\mathbf{x}_p) = \sum_{j=1}^{N} r_j \phi_j(\mathbf{x}_p) \quad ; \quad v(\mathbf{x}_p) = \sum_{i=1}^{N} \nu_i \phi_i(\mathbf{x}_p) \tag{1.20}$$

With some rearranging of the terms, the conservation criteria can now be rewritten as:

$$\sum_{j=1}^{N} r_j \underbrace{\left[ \sum_{p=1}^{N_p} \phi_j(\mathbf{x}_p) \phi_i(\mathbf{x}_p) \right]}_{A_{ij}} = \underbrace{\sum_{p=1}^{N_p} R_p \phi_i(\mathbf{x}_p)}_{b_i} \quad ; \quad \forall i = 1, ..., N \tag{1.21}$$

solving the above equation is equivalent to solving the linear system of equations (LSE)

$$\mathbf{A}\mathbf{r} = \mathbf{b} \tag{1.22}$$

$\mathbf{A} = \{A_{ij}\}_{i,j=1}^{N}$ : system matrix
$\mathbf{r} = \{r_j\}_{j=1}^{N}$ : correction field coefficients vector
$\mathbf{b} = \{b_i\}_{i=1}^{N}$ : right hand side (RHS)/load vector

The derivation shown above is similar to the classic Finite Element Method used to solve elliptic and parabolic PDEs. The LSE can be solved either directly (inversion) or using iterative solvers such as Jacobi, Gauss-Seidel or Conjugate Gradient methods. The corrected velocity field can finally be computed by subtracting the residual field and plugging in the found coefficients $r_j$:

$$\tilde{Q}_p^{VFP} = Q_p^{VFP} - \tilde{R}(\mathbf{x}_p) = Q_p^{VFP} - \sum_{j=1}^{N} r_j \phi_j(\mathbf{x}_p) \tag{1.23}$$

The momentum correction is the first step and allows the velocity field distribution to be shifted towards the correct mean velocity field. The second step is the energy correction which helps correct the random particle velocity fluctuations inside mesh cells; finally leading to a conservative system.

### 1.4.2 Energy correction

The energy correction step is applied after the velocity field has been corrected w.r.t momentum conservation. The energy correction consist of a cell specific correction term applied to the cells velocity fluctuations w.r.t cell mean velocity. The $i$-th cell ($\Omega_i$) mean velocity is given by ensemble averaging the particles velocities for all particles inside the cell:

$$\langle \tilde{Q}^{VFP} \rangle_i = \sum_{p=1:\ \mathbf{x}_p \in \Omega_i}^{N_p} \tilde{Q}_p^{VFP} \tag{1.24}$$

The quantities used are the corrected velocity components computed after the momentum correction step. The particles velocity fluctuation w.r.t mean are then defined as:

$$\tilde{q}_p^{VFP} = \tilde{Q}_p^{VFP} - \langle \tilde{Q}^{VFP} \rangle_i \ \ ; \ \ \forall p \ s.t \ \mathbf{x}_p \in \Omega_i \ \ ; \ \ \tilde{q}_p^{\bullet} \in \{\tilde{u}_{p,1}^{\bullet}, \tilde{u}_{p,2}^{\bullet}, \tilde{u}_{p,3}^{\bullet}\} \tag{1.25}$$

and the $p$-th particle kinetic energy in cell $i$ is:

$$\hat{K}_{p,i} = \frac{1}{2}[(\tilde{u}_{p,1}^{VFP})^2 + (\tilde{u}_{p,2}^{VFP})^2 + (\tilde{u}_{p,3}^{VFP})^2] \tag{1.26}$$

Using the same procedure used for the momentum correction, a kinetic energy residual field $\hat{R}(\mathbf{x})$ is computed:

$$\sum_{p=1:\ \mathbf{x}_p \in \Omega_i}^{N_p} [\hat{K}_{p,i} - \hat{R}(\mathbf{x}_p)] v(\mathbf{x}_p) = 0$$

Afterwards the conservation condition is rewritten as the multiplication of a cell scalar coefficient $\kappa_i$ to the cell total kinetic energy

$$\kappa_i^2 \sum_{p=1:\ \mathbf{x}_p \in \Omega_i}^{N_p} \hat{K}_{p,i} = \sum_{p=1:\ \mathbf{x}_p \in \Omega_i}^{N_p} [\hat{K}_{p,i} - \hat{R}(\mathbf{x}_p)] \tag{1.27}$$

the cell $\Omega_i$ correction coefficient can be computed with:

$$\kappa_i = \left[ 1 - \sum_{p=1:\ \mathbf{x}_p \in \Omega_i}^{N_p} \frac{\hat{R}(\mathbf{x}_p)}{\hat{K}_{p,i}} \right]^{1/2} \tag{1.28}$$

The final velocities are given by

$$Q_p^{VFP,corr} = \langle \tilde{Q}^{VFP} \rangle_i + \kappa_i \tilde{q}_p^{VFP} \ \ ; \ \ \forall p \ s.t \ \mathbf{x}_p \in \Omega_i \tag{1.29}$$

It should be noted that a multiplicative correction ($\kappa_i$) is used instead of an additive one, as the latter could lead to negative energies.

**Basis Function choice:** The basis functions $\phi(\mathbf{x})$ need to satisfy the cardinal basis property on a given mesh. The easiest choice is the hat/tent function defined on a tensor product mesh. Important to notice that other choices of basis functions could be possible, such as the higher order used in Lagrangian FEM.

## 1.5 Electromagnetic solver

In order to compute the particles velocities, we first need to compute the electromagnetic forces acting on them; i.e we need to compute the background electric and magnetic field.
In the current framework a static magnetic field (constant in time but not in space) is considered. The third Maxwell equation can thus be simplified to obtain an irrotational electric field:

$$\mathbf{B}(\mathbf{x},t) = \mathbf{B}(\mathbf{x}) = \mathbf{B}_{ext} \rightarrow \nabla_x \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} = 0 \tag{1.30}$$

This means that we are not considering self induced magnetic fields and we just use a constant external magnetic field that acts on the plasma. The assumptions made allows us to define a potential field $\varphi(\mathbf{x},t)$:

$$\mathbf{E} = -\nabla_x \varphi(\mathbf{x},t) \tag{1.31}$$

Inserting the above relation in Maxwell first equation

$$\nabla_x \mathbf{E} = \frac{\rho(\mathbf{x},t)}{\varepsilon_0} \tag{1.32}$$

leads to a Poisson like equation for the potential field

$$\nabla_x^2 \varphi(\mathbf{x},t) = -\frac{\rho(\mathbf{x},t)}{\varepsilon_0} \tag{1.33}$$

with vacuum permittivity $\varepsilon_0$ and charge density (defined in presence of multiple species)

$$\rho(\mathbf{x},t) = \sum_s q_s n_s(\mathbf{x},t) \tag{1.34}$$

In our framework we work with two species, namely electrons and positive ions. The charge density thus become

$$\rho(\mathbf{x},t) = q_{e-} n_{e-}(\mathbf{x},t) + q_{i+} n_{i+}(\mathbf{x},t)$$

The electrons and ions number densities are computed for each cell using the number of computational particles we are simulating. The value is then interpolated to the grid nodes (scattering) through the use of tent basis functions . The kinetic computation of number density at grid node $j$ for species $s$ is thus defined as [11]

$$n_s(\mathbf{x}_j,t) = \sum_{p=1:\ \mathbf{x}_p \in \Omega_i}^{N_{p,s}} \frac{w}{vol(\Omega_i)} \phi_j(\mathbf{x}_j - \mathbf{x}_p) \quad ; \ \forall i = 1,...,N_{cells} \quad ; \quad \forall j: \ \mathbf{x}_j \in \Omega_i \tag{1.35}$$

$N_{p,s}$ : total number of particles of species $s$
$N_{cells}$ : total number of cells

The relative low mass and velocity of electrons allows us to simplify computations by computing the electrons number density through the Boltzmann relation [7]:

$$n_{e-}(\varphi(\mathbf{x},t)) = n_{e-}(\varphi_0(\mathbf{x},t)) \exp\left[q_{e-} \frac{(\varphi(\mathbf{x},t) - \varphi_0(\mathbf{x},t))}{k_b T_{e-}}\right] \tag{1.36}$$

with reference potential and number density $\varphi_0(\mathbf{x}, t)$, $n_{e-}(\varphi_0(\mathbf{x}, t))$ respectively. Comparative simple test cases where the electron number density is computed using the two shown method are presented in chapter 4. In the considered framework, the Poisson eqn. (1.33) is discretized using a simple, yet effective, second order finite difference scheme. Once the potential field and the respective electric field has been computed using charge density evaluated at the grid nodes at the specified time step, the data is linearly interpolated back from grid nodes to the particles (gathering).

# 2 Code developement

## 2.1 Collisionless Plasma

### 2.1.1 Full kinetic electrostatic test case

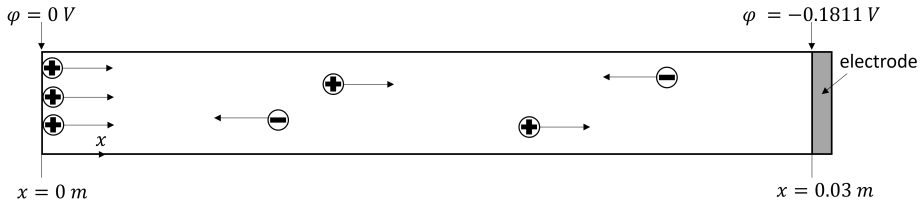


**Figure 2.1:** Sketch of the 1-dimensional Plasma Sheath domain configuration. Particles injected at the left boundary and move towards the electrode at the right boundary. Mesh size $\Delta x$ corresponds to the Debye length and the time step size is chosen using a CFL condition. Showed in the sketch are also the applied Neumann and Dirichlet BCs.

The full kinetic simulation of the Vlasov equation has been implemented and tested on the 1-dimensional plasma sheath test case. The Vlasov eqn. is solved using the Boris scheme [8]. Particles are injected from the left boundary towards the right boundary. At the right boundary an electrode with negative potential is present. Positive ions are attracted to the electrode and will accumulate on it. The accumulation of positive ions will repel electrons from the electrode, making them move towards the left boundary. The electric field is static across the entire domain. For this test case exists a closed form solution derived from the Boltzmann relation [7]; which can be used to compare the results. The closed form solution for the electric potential is solved using forward finite difference and the resulting equation reads:

$$\frac{\varphi^{j+1} - \varphi^j}{\Delta x} = f(\varphi^j) = -\frac{k_b T_{e-}}{q_{e-}} \left[ 2\theta^2 \left( \left( 1 - \frac{2 q_{e-} \varphi^j}{k_b T_{e-} \theta^2} \right)^{1/2} - 1 \right) + 2 \exp\left( \frac{q_{e-}}{k_b T_{e-}} \varphi^j \right) - 2 \right]^{1/2}$$

$\theta = U_{i+} / \sqrt{k_b T_{e-} / m_{i+}}$
$U_{i+}$ : ions injection velocity
$m_{i+}$ : ions (individual particles) mass
$T_{e-}$ : electrons temperature
$q_{e-}$ : electrons charge
$\Delta x$ : mesh size

### 2.1.2 Plasma flow around an obstacle

In real world applications we are interested in simulating the flow around objects with complex geometries inside the computational domain. As a first step towards these type of simulations,

a test case with rectangular object (obstacle) inside the domain as shown in Fig. 2.2 had previously been developed. Comparisons with the reference (not used as physical reference but as algorithmic reference instead) case produced by the specialized plasma simulation company PIC Consulting LLC [4] of the same set-up showed that the our framework was predicting the incorrect particle around the conductive obstacle as shown in Fig. 2.3 (reference) and 2.4 (our case). The reference plot for the density (Fig. 2.3 ) indicates that particles trajectories should bend much more than what we observed (Fig. 2.4). In the considered case a collisionless plasma described by the Vlasov eqn. is solved using the Boris algorithm [8].

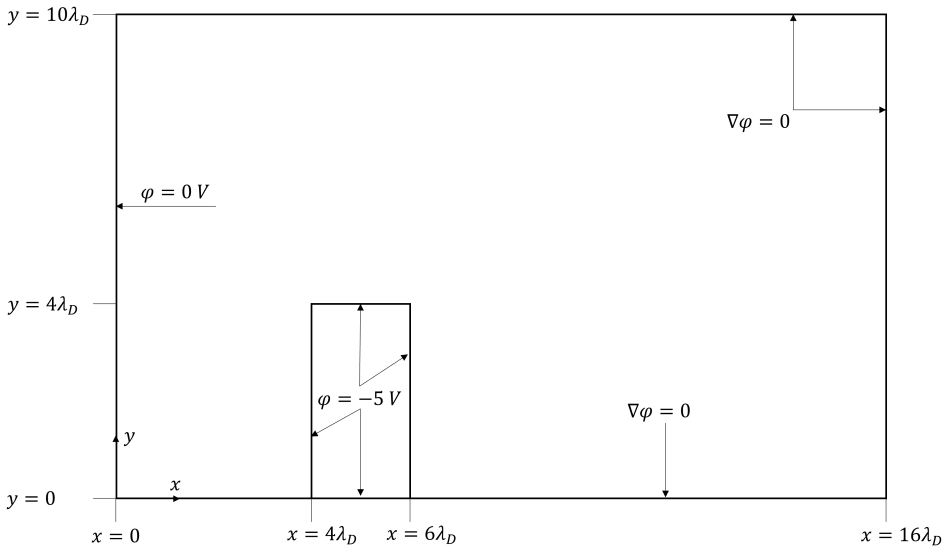


**Figure 2.2:** Sketch of the 2-dimensional Plasma flow around an obstacle domain configuration. Particles injected at the left boundary. All domain boundaries are open for the particles, except the bottom boundary which has a reflective BC on it. Mesh size $\Delta x$ corresponds to the Debye length and the time step size is chosen using a CFL condition. The negative potential imposed at the obstacle boundary will attract the positively charged particles, leading to a bending of the flow. Showed in the sketch are also the applied Neumann and Dirichlet BCs.

In addition, when in presence of a fully reflective object, the reflective boundary conditions would be applied to particles away from the obstacle, leading to particles bouncing on an "invisible wall" at the obstacle location as shown in Fig 2.5. In absence of obstacles the computations were correct; therefore a bug (or more) had to be present somewhere in the code related to the object boundary handling. In the next two subsections the root causes of the observed problems is explained.

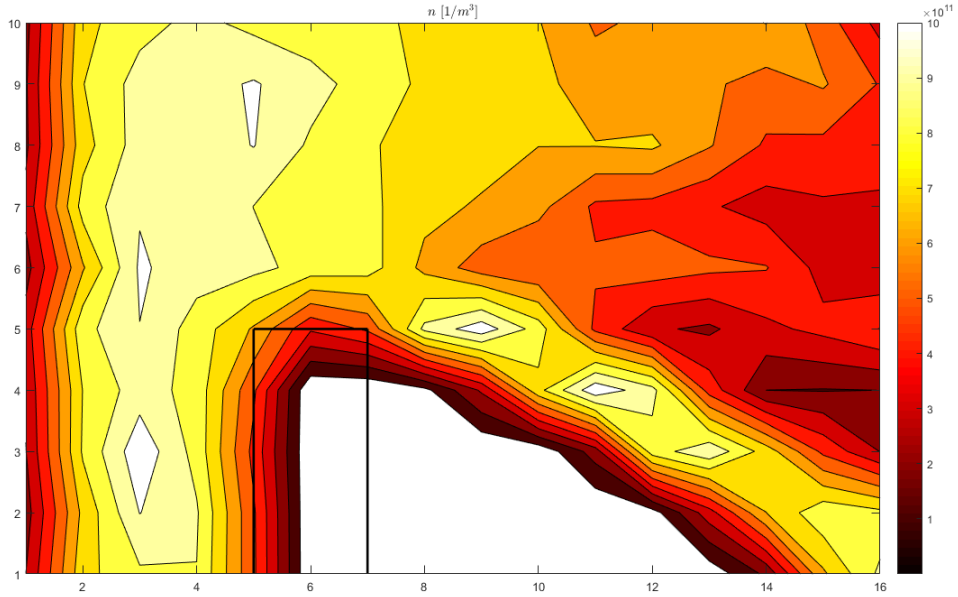


**Figure 2.3:** Plot of particles number density produced by the reference implementation [4]

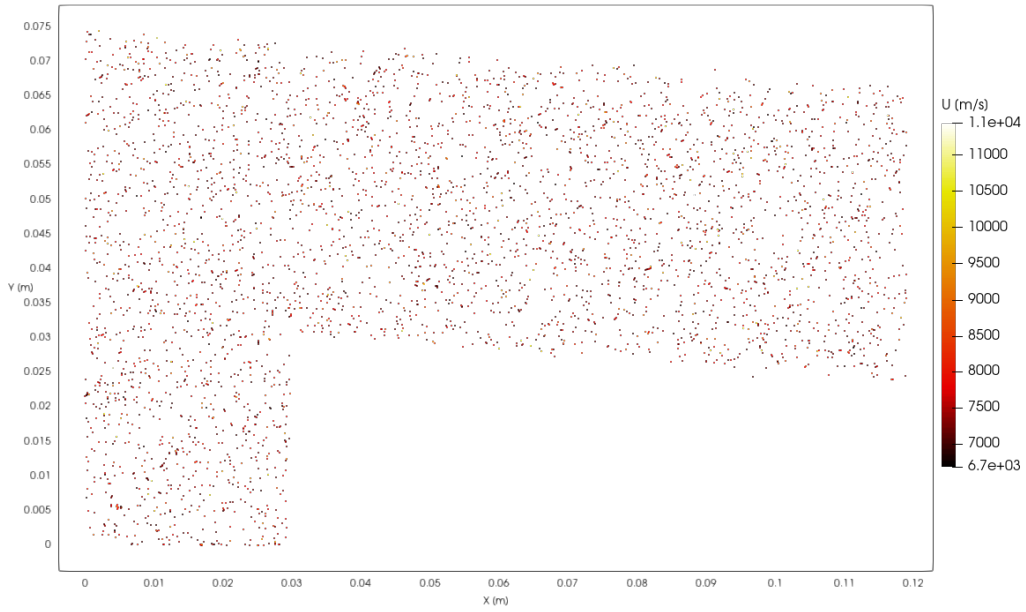


**Figure 2.4:** Plot of particles position colored with the longitudinal velocity produced by our framework before bugs were fixed. The obstacle has a conductor boundary condition, i.e charges are allowed to accumulate on it

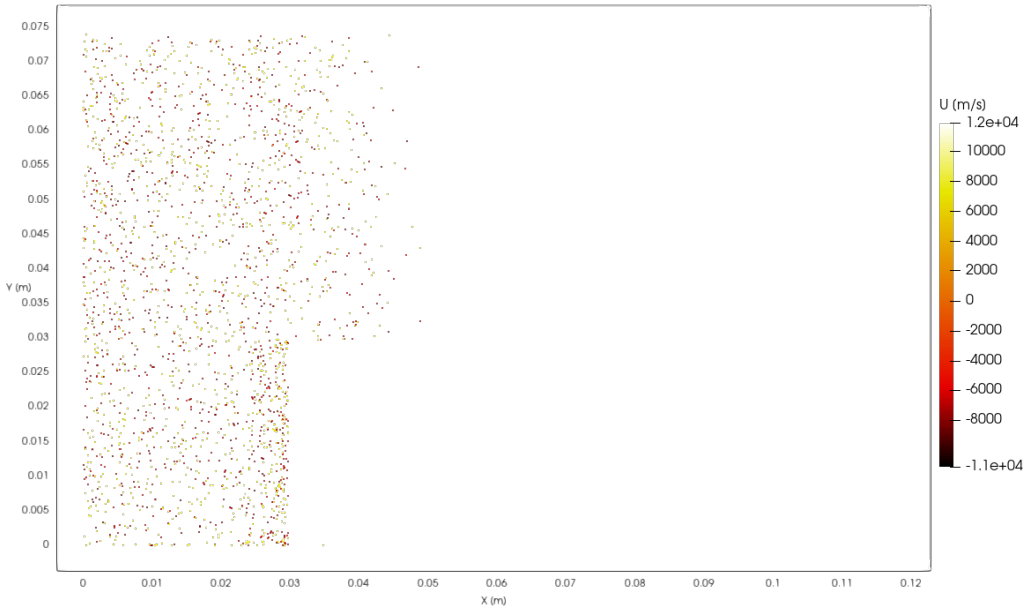


**Figure 2.5:** Plot of particles position colored with the longitudinal velocity produced by our framework before bugs were fixed. The obstacle has a reflective boundary condition, i.e particles that impact on the obstacle are reflected in specular manner w.r.t it.

### Particles boundary detection

If a particle enters an object by crossing a boundary wall during the timestep position update, we apply the specific wall BC to the interested particles. While investigating the root cause of the problem appearing when inserting obstacles in the domain, we noticed that function responsible for checking which particle have crossed a wall could create problems in presence of a fully reflective (boundary condition wise) object and the implementation was not optimal. A solution has been found and correctly implemented. It was discovered afterwards that the boundary crossing detection was not actually the only cause of the observed incorrect potential field.

In the previous implementation the distance between current and next particle position $d_p$, and between boundary wall and current position $d_w$ was computed. Afterwards a ratio $r$ between the two distances would be computed:

$$r = \frac{d_w}{d_p}$$

A ratio $r < 1$ would indicate that a particle might be crossing the wall. The distances used were aligned with the axis on which the considered boundary was applied (e.g boundary on $x-$, then use $x$ distance).

The next step was to check that the particle was effectively travelling towards the wall. This was done by checking the direction of travel w.r.t the wall by looking at sign of the dot product between wall normal vector $\mathbf{n}_{BC}$ and particle velocity vector $\mathbf{u}$. A particle would cross a wall if

$r < 1$ and $sign(\langle \mathbf{u}, \mathbf{n}_{BC} \rangle) < 1$ (i.e moving towards wall). One of the issue with this implementation was the wall point $B$ used to get the position vector w.r.t the wall. This point was defined as either the object origin vertex or the vertex furthest away from the object origin. Those points were used because depending on which boundary side was selected between x+,x-,y+,y- (z+,z- in 3D) to apply the BC on; the chosen point $B$ would always lie on the the correct selected boundary line (or plane in 3D). Overall this method would not behave correctly. A 2D sketch of the boundary detection algorithm with the definition of the vectors can be seen in Fig 2.6. This method is sub-optimal since, as seen in the simulations, particles might be wrongly flagged as boundary crossing.

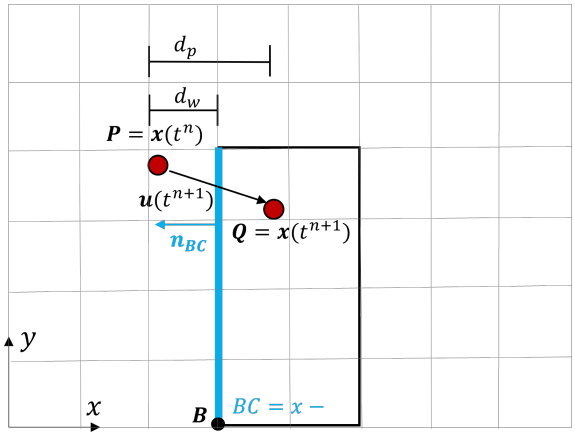


**Figure 2.6:** Sketch of the obstacle case. Shown is a specific particle configuration and all quantities used in the old implementation of the boundary crossing detection algorithm

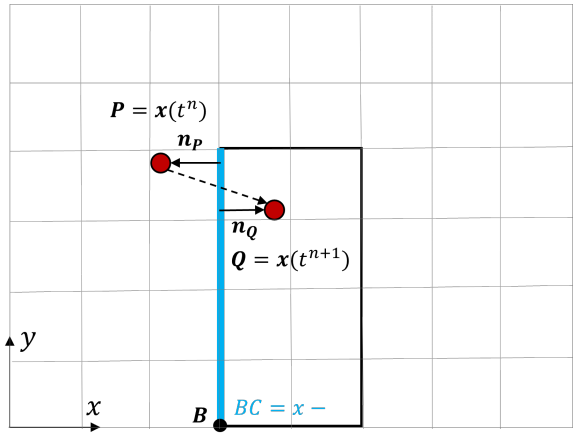


**Figure 2.7:** Sketch of the obstacle case. Shown is a specific particle configuration and all quantities used in the new implementation of the boundary crossing detection algorithm.

In the updated implementation the first step consist of computing the (normal) vectors between current particle position and boundary wall $\mathbf{n_P}$, and between next particle position and the wall $\mathbf{n_Q}$. Again, only the vector component aligned with the axis which has the BC is considered. Afterwards the dot product between the two vector is computed; a negative value signifies that the particle has crossed the wall. A 2D sketch of the algorithm can be seen in Fig. 2.7. Additionally, in Fig. 2.8, the sketch of a situation where the old boundary detection algorithm would fail while the new proposed method would not fail is presented.
It is clear that the new approach completely avoids the issue. From the same figure it is also noticeable the only limitation of the new approach; i.e when particles trajectories intersect perfectly the obstacle vertex. In those rare scenarios, the code (due to the boundary loop order) would reflect the particle w.r.t the lower or upper horizontal (y-,y+) obstacle boundaries (depending on the considered vertex).

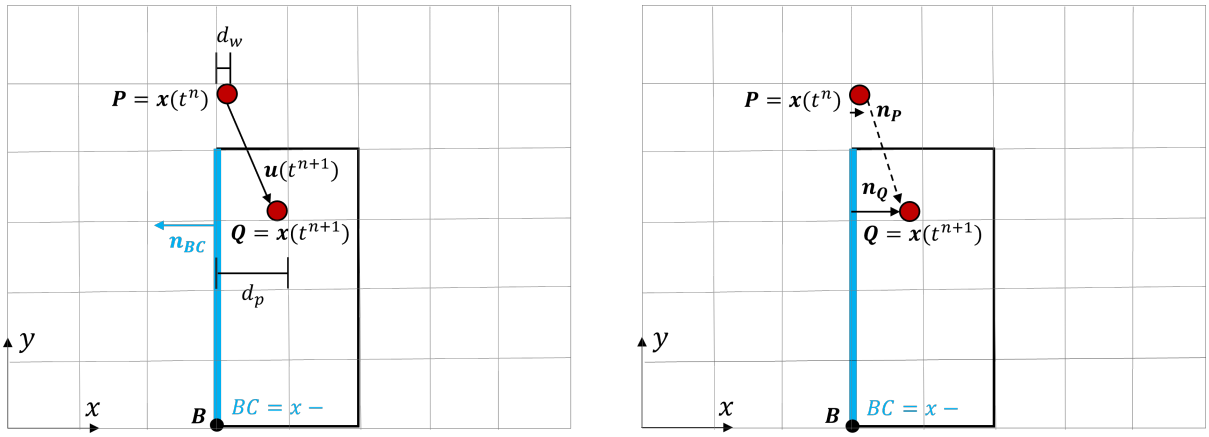


**Figure 2.8:** Sketch of the obstacle case. Shown is a specific particle configuration which would induce the old implementation of the boundary crossing detection algorithm to fail (left) and the new implementation (right) which can handle the specific case

### Post-processing

The incorrect predicted solutions shown Figs. 2.4, 2.5 were not caused only by a wrong boundary wall detection but also by an incorrect output of the post-processing data , which would lead to a portion of particles data not being outputted (therefore it seemed like those particle did not exist).

Other components of the code were tested and scrutinized before (on the way to) finding this bug. Therefore, even if the fix was easy, the journey to find the bug has allowed us to thoroughly evaluate/test the code components (boundaries, electromagnetic solver, particle pusher).

## 2.2 Collisional Plasma

### 2.2.1 Sparse LSE iterative solvers

The momentum and energy correction are applied after each time steps. The LSE system matrix scales with the total number of grid points and requires to store $N^2$ entries assuming we have a grid with $N$ nodes. E.g in a 2-dimensional grid with $N_i = N_j = 1000$ nodes in each dimension ($N = N_i N_j$), assuming double precision, the system matrix requires to store 8000 GB in memory; if we also consider the cost of constructing upper, lower and diagonal matrices for iterative solvers, the cost increases even more. The excessive memory cost can be reduced by employing sparse matrix representations (COO, CRS,...) and then utilize sparse LSE (iterative) solvers. Our framework has been written using the Python library PyTorch in order to allow GPU acceleration of certain computations. PyTorch fully supports sparse matrix representation but the provided iterative solvers either do not support sparse representations or are not actually operating on sparse matrices (i.e they reconstruct the full matrices from the sparse ones) [9]. For this reason, custom sparse iterative solvers have been implemented in our framework.

For a given $i$-th row of the system matrix $\mathbf{A}$, the non-zero entries locations correspond to the indices of the nodes contained in the support of the basis function evaluated at the (diagonal) $i$-th node. E.g for a 2-dimensional grid such as the one showed in Fig. 2.9, when evaluating the tent basis function at node $i$, non-zero entries will be induced by the two adjacent nodes on the same grid row and from the nodes on the lower and upper grid row. Therefore for a generic

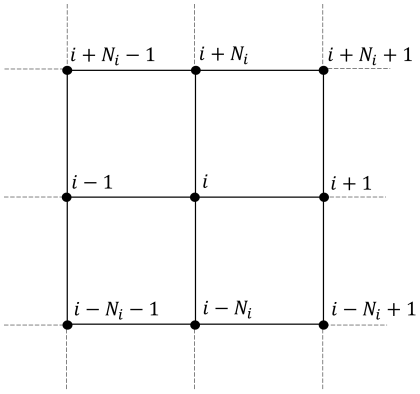


**Figure 2.9:** Sketch of a generic 2-dimensional orthogonal grid section. Shown are all the nodes part of the support of the basis function $\phi(x_i)$. $N_i$ represent the number of nodes in $x$ direction.

$i$-th grid node the system matrix $i$-th row will have the structure (each non-zero matrix entry is written as $A_{ij}$, $i, j = 1, ..., N_i N_j$):

$$\underbrace{0 \cdots \; A_{i,i-N_i-1} \; A_{i,i-N_i} \; A_{i,i-N_i+1} \; 0 \cdots 0 \; A_{i,i-1} \; A_{i,i} \; A_{i,i+1} \; 0 \cdots 0 \; A_{i,i+N_i-1} \; A_{i,i+N_i} \; A_{i,i+N_i+1} \cdots \; 0}_{N_i N_j \text{ elements}}$$

If the $i$-th node lies on a boundary then the respective adjacent entries in the matrix will be zero. In 2D we can therefore construct the following $N_i N_j \times 1$ vectors and $N_i N_j \times 3$ matrices to store the non zero elements:

$$\mathbf{D} = \{A_{i,i}\}_{i=1}^{N_i N_j} \quad ; \quad \mathbf{L_x} = \{A_{i,i-1}\}_{i=1}^{N_i N_j} \quad ; \quad \mathbf{U_x} = \{A_{i,i+1}\}_{i=1}^{N_i N_j}$$

$$\mathbf{L_y} = \{(A_{i,i-N_i-1}, A_{i,i-N_i}, A_{i,i-N_i+1})\}_{i=1}^{N_iN_j} \quad ; \quad \mathbf{U_y} = \{(A_{i,i+N_i-1}, A_{i,i+N_i}, A_{i,i+N_i+1})\}_{i=1}^{N_iN_j}$$

Assuming again $N_i = N_j = N = 1000$ and double precision, the above tensors require 72 MB of storage. The symmetry of the system matrix is also used to reduce the number of iterations (loops) needed to compute all entries and assign them to the respective container:

$$A_{ij} = \sum_{p}^{N_p} \phi_j(x_p)\phi_i(x_p) = A_{ji}$$

The constructed data container are then utilized by the iterative solver to compute the LSE solution.

### Jacobi iterative solver

The classic Jacobi method applied to our LSE reads:

$$r_i^{(k+1)} = -\frac{1}{A_{i,i}} \left( \sum_{j \neq i} A_{i,j} r_j^{(k)} - b_i \right)$$

Due to the special structure of our system matrix, we can rewrite the iterative solution $i$-th component as:

$$\begin{aligned} r_i^{(k+1)} = -\frac{1}{A_{i,i}} \Big( & A_{i,i-N_i-1} r_{i-N_i-1}^{(k)} + A_{i,i-N_i} r_{i-N_i}^{(k)} + A_{i,i-N_i+1} r_{i-N_i+1}^{(k)} + A_{i,i-1} r_{i-1}^{(k)} \\ & + A_{i,i+1} r_{i+1}^{(k)} + A_{i,i+N_i-1} r_{i+N_i-1}^{(k)} + A_{i,i+N_i} r_{i+N_i}^{(k)} + A_{i,i+N_i+1} r_{i+N_i+1}^{(k)} - b_i \Big) \end{aligned}$$

It is now clear the utility of the previously defined data container as we can easily compute all entries of the solution vector as a sum of element-wise vector multiplications (noted with the Hadamard product $\circ$):

$$\begin{aligned} \mathbf{r}^{(k+1)} = -D^{-1} \circ \Big( & \mathbf{L_{y,1}} \circ \mathbf{r}_{-N_i-1}^{(k)} + \mathbf{L_{y,2}} \circ \mathbf{r}_{-N_i}^{(k)} + \mathbf{L_{y,3}} \circ \mathbf{r}_{-N_i+1}^{(k)} + \mathbf{L_x} \circ \mathbf{r}_{-1}^{(k)} \\ & + \mathbf{U_x} \circ \mathbf{r}_{+1}^{(k)} + \mathbf{U_{y,1}} \circ \mathbf{r}_{+N_i-1}^{(k)} + \mathbf{U_{y,2}} \circ \mathbf{r}_{+N_i}^{(k)} + \mathbf{U_{y,3}} \circ \mathbf{r}_{+N_i+1}^{(k)} - \mathbf{b} \Big) \end{aligned}$$

with $(\mathbf{L}, \mathbf{U})_{\mathbf{y,1}} = \{A_{i,i\mp N_i-1}\}_{i=1}^{N_iN_j}$, $(\mathbf{L}, \mathbf{U})_{\mathbf{y,2}} = \{A_{i,i\mp N_i}\}_{i=1}^{N_iN_j}$, $(\mathbf{L}, \mathbf{U})_{\mathbf{y,3}} = \{A_{i,i\mp N_i-1}\}_{i=1}^{N_iN_j}$ and $\mathbf{r}_{\bullet}^{(k)}$ being equivalent to the $\mathbf{r}^{(k)}$ vector with elements positions shifted by $\bullet$ amount in a periodic manner.

These computations can now be easily parallelized by the GPU. The periodic shifting of the vector $\mathbf{r}^{(k)}$ entries leads to the boundary entries of the vector $\mathbf{r}^{(k+1)}$ to be wrongly computed. For this reason the first and last $N_i$ (assuming 2-dimensional grid) entries have to be "manually" computed (element by element) through a loop. Overall the fraction of manually computed entries is $2N_i/N_iN_j = 2/N_j$. This fraction represent less than one quarter of the total entries for all grids with $N_j > 8$, i.e is negligible for most real use cases. A different implementation of the boundary term computation might enable to avoid using a loop for the boundary terms computations.

### Conjugate gradient iterative solver

Similarly, we can utilize the same approach to simplify other iterative solvers such as Conjugate gradient. The simplification occurs when computing the matrix vector multiplication between the LSE system matrix $\mathbf{A}$ and the conjugate vectors $\mathbf{p}_k$. For the $i$-th elements the multiplication reads:

$$(\mathbf{A}\mathbf{p}_k)_i = \sum_{j=1}^{N_i N_j} A_{ij}\mathbf{p}_j^{(k)}$$

By expanding the sum and removing the zero entries we get:

$$(\mathbf{A}\mathbf{p}^{(k)})_i = A_{i,i-N_i-1}p_{i-N_i-1}^{(k)} + A_{i,i-N_i}p_{i-N_i}^{(k)} + A_{i,i-N_i+1}p_{i-N_i+1}^{(k)} + A_{i,i-1}p_{i-1}^{(k)}$$

$$+A_{i,i}p_i^{(k)} + A_{i,i+1}p_{i+1}^{(k)} + A_{i,i+N_i-1}p_{i+N_i-1}^{(k)} + A_{i,i+N_i}p_{i+N_i}^{(k)} + A_{i,i+N_i+1}p_{i+N_i+1}^{(k)}$$

All the entries of the matrix vector multiplication can be computed all at once on the GPU using the custom defined data container:

$$\mathbf{A}\mathbf{p}^{(k)} = \mathbf{L}_{\mathbf{y},\mathbf{1}} \circ \mathbf{p}_{-N_i-1}^{(k)} + \mathbf{L}_{\mathbf{y},\mathbf{2}} \circ \mathbf{p}_{-N_i}^{(k)} + \mathbf{L}_{\mathbf{y},\mathbf{3}} \circ \mathbf{p}_{-N_i+1}^{(k)} + \mathbf{L}_{\mathbf{x}} \circ \mathbf{p}_{-1}^{(k)} + \mathbf{D} \circ \mathbf{p}^{(k)}$$

$$+\mathbf{U}_{\mathbf{x}} \circ \mathbf{r}_{+1}^{(k)} + \mathbf{U}_{\mathbf{y},\mathbf{1}} \circ \mathbf{r}_{+N_i-1}^{(k)} + \mathbf{U}_{\mathbf{y},\mathbf{2}} \circ \mathbf{r}_{+N_i}^{(k)} + \mathbf{U}_{\mathbf{y},\mathbf{3}} \circ \mathbf{r}_{+N_i+1}^{(k)}$$

The definition of the coefficients follows the same rule as seen in the Jacobi method. Also for Conjugate gradient the resulting vector boundary terms have to be manually computed due to the vector $\mathbf{p}^{(k)}$ index shifting.

# 3 Results

## 3.1 Collisionless Plasma

### 3.1.1 Full kinetic electrostatic test case

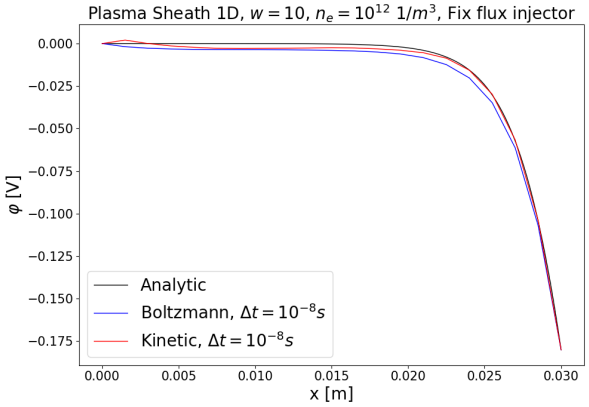


**Figure 3.1:** Plot of the electric potential predicted for the 1-dimensional Plasma Sheath test case in presence of a fix-flux injector/inflow BC. Plotted are three different solutions.

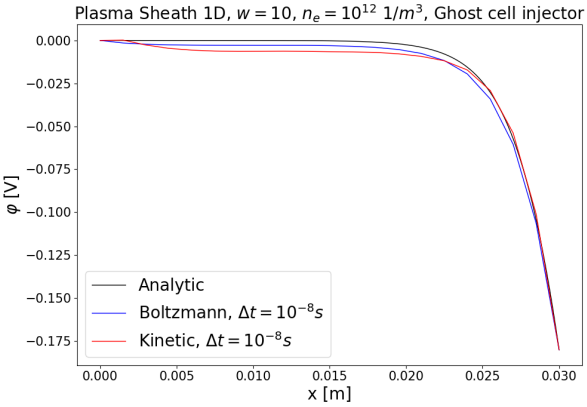


**Figure 3.2:** Plot of the electric potential predicted for the 1-dimensional Plasma Sheath test case in presence of a ghost-cell injector/inflow BC. Plotted are three different solutions

The 1-dimensional Plasma Sheath case has been used to compare the fully kinetic approach to a partially kinetic approach employing the Boltzmann relation simplification. Two different inflow boundary conditions were utilized (fix flux injector Fig. 3.1 and ghost cell injector Fig. 3.2). The difference between the two boundary conditions is how the particles initial velocity is initialized, where on the cell surfaces the particles are injected and how many particles are injected per time step. A statistical weight of 10 was used and the species present are electrons and Argon ions. For both inflow BCs, it can be seen that the fully kinetic approach produces a solution which agrees with the analytic one; similarly for the Boltzmann relation simplified simulation. One of the advantage brought by Boltzmann relation is the reduced computational time (since no gathering-scattering is made for electrons); on average the fully kinetic simulation took $\approx 1200s$ while the Boltzmann simulation took $\approx 400s$ on an Intel i7 4790K 4.4 GHz CPU.

### 3.1.2 Plasma flow around an obstacle

Fig. 3.3 shows the obstacle test case in presence of a reflective obstacle after the correction of the boundary crossing detection algorithm. It can be seen how particles are able to flow over the obstacle without any problem and being correctly reflected when impacting on it. Fig. 3.4 shows the obstacle case in presence of a conductive obstacle (i.e charges accumulate on it) after both the boundary detection bug and post-processing output data bug have been fixed. The showed behaviour is in line with the reference simulation used. It is important to note that the reference simulations are used just to validate the code and not as true physically correct references.

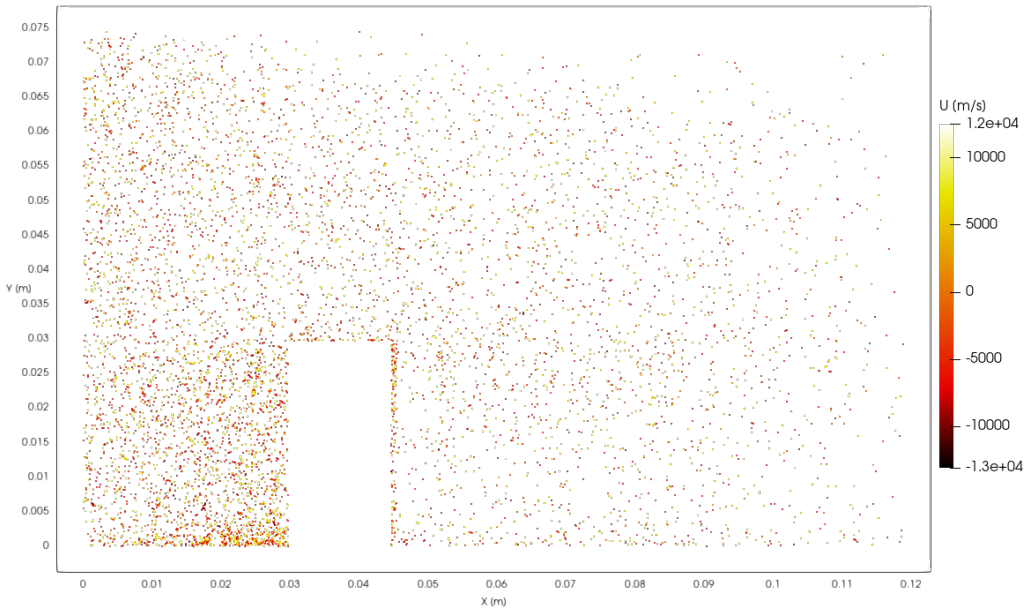


**Figure 3.3:** Plot of particles position colored with the longitudinal velocity produced by our framework after bugs were fixed. The obstacle has a reflective boundary condition.

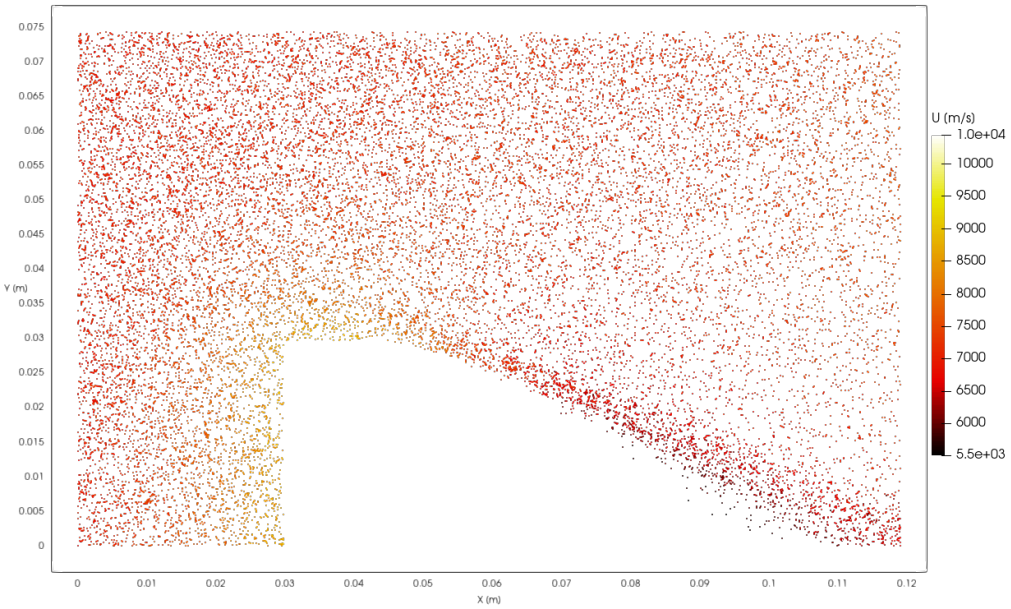


**Figure 3.4:** Plot of particles position colored with the longitudinal velocity produced by our framework after bugs were fixed. The obstacle has a conductor boundary condition

## 3.2 Collisional Plasma

### 3.2.1 Sparse LSE iterative solvers

The implemented sparse iterative solvers have been validated w.r.t dense solvers and direct inversion method. The L1 norm between the solution vector produced by different method was observed to approach the null vector, i.e the results are identical. The accuracy of the output depends a lot on the tolerance used for both Jacobi and Conjugate gradient. The tests were conducted on a simplified VF-Langevin velocity equation in 2 dimensions:

$$Q_p^{VFP} = Q_p^V + A\Delta t + B\xi$$

with constant coefficients A,B and uniformly distributed random number $\xi$. The notation is the same used in the Momentum correction section. A correct sparse solver implementation has to produce a corrected collision velocity field which is globally conserved. As we can see from Fig. 3.5 the sum over the particles of corrected velocity (red) in fact approaches zero after correction. The figure shows the exemplary results produced by the Conjugate Gradient solver.

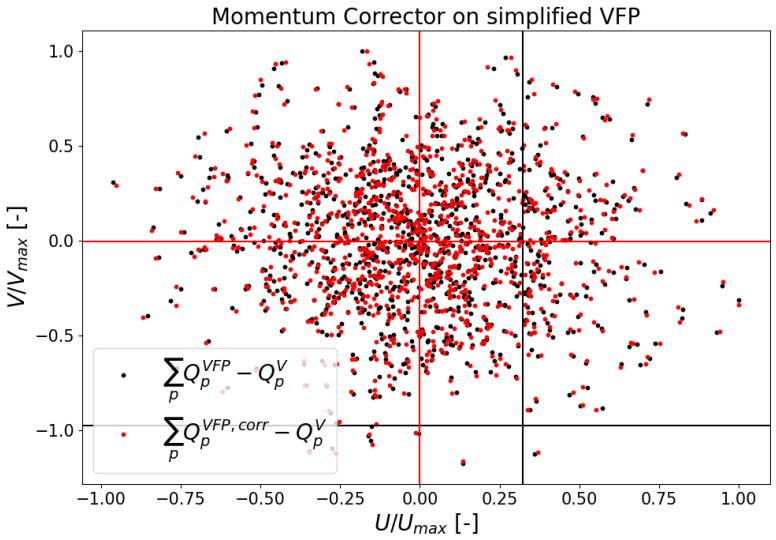


**Figure 3.5:** Plot of dimensionless velocities in U,V directions. Straight lines indicate the global momentum conservation value. Iterative solver used is Conjugate Gradient

# 4 Conclusions

The magnetised Poisson-VFP equation provides a simplified model to simulate the complex plasma behaviour, while still accounting for collisional kinetics and electromagnetic interactions. In this work, tools to enable the full simulation of the equation have been tested, fixed and further developed.

# Bibliography

[1] P. Jenny, H. Gorji, Accurate particle time integration for solving Vlasov-Fokker-Planck equations with specified electromagnetic fields, J. Comput. Phys. (2019), https://doi.org/10.1016/j.jcp.2019.02.040

[2] H. Gorji, P. Jenny, Fokker–Planck–DSMC algorithm for simulations of rarefied gas flows, J. Comput. Phys. (2015), https://doi.org/10.1016/j.jcp.2015.01.041

[3] P. Jenny *et al*, A solution algorithm for the fluid dynamic equations based on a stochastic model for molecular motion, J. Comput. Phys. (2019), https://doi.org/10.1016/j.jcp.2009.10.008

[4] Particle in Cell Consulting LLC, Simple Particle in Cell Code in Matlab, (2011), https://www.particleincell.com/2011/particle-in-cell-example/, accessed 16.11.2022

[5] M.N. Rosenbluth *et al*, Fokker-Planck Equation for an Inverse-Square Force, Physical Review (1957), https://doi.org/10.1103/PhysRev.107.1

[6] Claus-Dieter Munz *et al*, Coupled Particle-In-Cell and Direct Simulation Monte Carlo method for simulating reactive plasma flows, Physical Review (2014), https://doi.org/10.1016/j.crme.2014.07.005

[7] M. Pfeiffer *et al*, A Particle-in-Cell solver based on a high-order hybridizable discontinuous Galerking spectral element method on unstructured curved meshes, Comput. Methods Appl. Mech. Engrg (2019), https://doi.org/10.1016/j.cma.2019.02.014

[8] Hong Quin *et al*, Why is Boris algorithm so good?, Physics of Plasmas (2013), http://dx.doi.org/10.1063/1.4818428

[9] Meta AI, PyTorch Documentation, https://pytorch.org/docs/stable/sparse.html, accessed 23.12.2022

[10] P. Jenny, Advanced CFD Methods lecture notes, ETH Zürich, (2022)

[11] C.K. Birdsall, A.B Langdon, Plasma Physics via Computer Simulation, Series in Plasma Physics (1991), https://doi.org/10.1201/9781315275048

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

## Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

---

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

**Title of work** (in block letters):

IMPLEMENTATION AND VALIDATION OF THE MAGNETIZED POISSON-VLASOV-FOKKER-PLANCK FRAMEWORK

**Authored by** (in block letters):

*For papers written by groups the names of all authors are required.*

| **Name(s):** | **First name(s):** |
|---|---|
| FABIANO | SASSELLI |
| | |
| | |
| | |
| | |

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

| **Place, date** | **Signature(s)** |
|---|---|
| 23.12.2022 | Fabiano Sasselli |
| | |
| | |
| | |
| | |

*For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.*