# Non-Uniform Splines for Semi-Lagrangian Kinetic Simulations of the Plasma Sheath

Emily Bourne, Yann Munschy, Virginie Grandgirard, Michel Mehrenberger, Philippe Ghendrih

# Non-Uniform Splines for Semi-Lagrangian Kinetic Simulations of the Plasma Sheath

Emily Bourne[1], Yann Munschy[1], Virginie Grandgirard[1], Michel Mehrenberger[2], and Philippe Ghendrih[1]

[1]CEA, IRFM, Saint-Paul-les-Durance, F-13108, France
[2]I2M, CMI, UMR 7373, 39 rue Frédéric Joliot Curie, 13453 Marseille Cedex 13 France

August 9, 2022

**Abstract**

Numerical methods based on non-uniform splines of varying degrees are used to run kinetic semi-Lagrangian simulations of the plasma sheath. The sheath describes a region of plasma in contact with a wall, acting as a heat, momentum, and particle bath. The latter regulates the current loss and consequently the electric field. At micro scale it then influences the development of plasma turbulence and at a larger scale plasma rotation. However this region is particularly difficult to simulate, due to its kinetic nature and the presence of steep gradients requiring fine numerical resolution. In this work, a new well-conditioned method for determining the coefficients for Schoenberg's "best" quadrature is presented, providing improved precision compared to a naive approach based on b-spline derivatives (4 extra significant figures of precision are obtained for coefficients of the fifth order scheme). The construction of a simulation grid from Greville abcissae of judiciously chosen non-uniform knots is shown to reduce the memory requirements by 89% for the simulation presented. Cubic splines are found to provide a good compromise between fast convergence and controlled oscillations preserving positivity. Despite the reduction in performance associated with using non-uniform spline schemes, effective parallelisation through the use of GPUs leads to non-uniform simulations running 5.5 times faster than uniform simulations providing equivalent results.

## 1 Introduction

The plasma sheath is the part of a hot plasma adjacent to a comparatively cold wall. In this area there are steep gradients [1], for example in temperature, and density, compared to the rest of the plasma [2]. These constraints make this area particularly difficult to simulate. The behaviour in this area is inherently kinetic since particle velocity plays a strong role in establishing the plasma current. Furthermore, electrons and ions, having different average velocities, exhibit distinct behaviour. It is this difference in behaviour which leads to the formation of the sheath; therefore each species must be modelled individually. This precludes the use of simpler models such as fluid models and forces the use of more complicated kinetic models.

The main kinetic methods available for simulating this region are PIC (particle in cell) methods and Eulerian methods. The low density in the sheath imposes challenges for PIC methods as a very large number of particles must be simulated in order to obtain reasonable accuracy in the low-density regions. Similarly, steep gradients pose problems for Eulerian methods as a sufficient number of points must be used to sufficiently describe the slope. In a typical sheath simulation, the steep gradients occur over a distance which is smaller than the Debye length, while the simulation domain ought to be up to one million Debye lengths long. If too few sampling points are used then the areas of steep gradients can appear as discontinuities in the simulation. Such discontinuities can propagate over time causing non-physical behaviour often resulting from oscillations. It is therefore important to resolve this area with a spatial resolution smaller than the Debye length.

[1]Corresponding author : emily.bourne@cea.fr

When using equidistant sampling points, such a small resolution comes with a very large cost both in computation and memory consumption. This is especially the case for large multi-dimensional simulations such as the GYSELA code [3] which simulates the entirety of a tokamak in five dimensions. Such simulations already operate at the bounds of what is possible on today's supercomputers. It is therefore not feasible for these simulations to drastically increase the number of points used in the simulation. Non-equidistant sampling points allow the steep gradient to be resolved without unduly increasing the memory required for the simulation. This can cause significant memory gains in a five dimensional simulation as the removal of just one point in a given dimension, reduces the data required by $n_2 n_3 n_4 n_5$ points, where $n_2$, $n_3$, $n_4$, and $n_5$ are the number of points in the other four dimensions.

On the other hand, non-equidistant points can potentially increase the calculation requirements, as they require more complex algorithms in which the varying spatial resolution is calculated and handled appropriately. The choice between equidistant and non-equidistant points is therefore a trade-off between memory and computational requirements and is dependent upon the machine on which the calculations are run. Specifically the availability of GPUs can have a noticeable effect on this balance.

In this paper we will investigate the changes necessary to move from a semi-Lagrangian simulation on equidistant points to one with non-equidistant points, with a view to extending the GYSELA code [3] to more accurately simulate the edge domain of a tokamak plasma. Given the complexity of the GYSELA code, a reduced model based on the same algorithms is used to investigate this problem and determine whether the memory-calculation trade-off is acceptable. This reduced model will consider only the sheath, which is a complicated problem in its own right having been investigated in multiple other papers [2, 4, 5, 6]. The handling of the wall in this area as well as steep gradients in the sheath region itself mean that this simulation is highly pertinent for the testing of non-equidistant points, in addition to actually describing the region that will be added to GYSELA.

Previous works simulating the sheath with non-uniform points have used finite volumes methods [2, 4]; however this gives rise to time and spatial step restrictions. The GYSELA code [3] takes a different approach using a semi-Lagrangian method to reduce these restrictions. This is crucial in such a large code in order to facilitate running simulations over large time scales. Previous works using a semi-Lagrangian method for sheath simulations include the work of Badsi et al. [6]. They use Lagrange interpolation on uniform points to reconstruct the function, whereas in this paper, as in the GYSELA code [3], a spline interpolation will be used. Manfredi et al. [5] also studied the sheath on uniform points, using a semi-Lagrangian method, additionally using a slope corrector to ensure the positivity of the distribution function. Semi-Lagrangian advection on cubic splines with non-equidistant points in the velocity dimension has previously been investigated by Afeyan et al. in a different context: the study of KEEN waves [7].

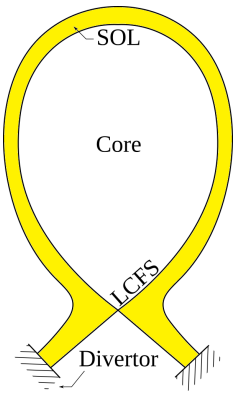
This work distinguishes itself from previous work by the use of non-equidistant points in all dimensions, Greville abcissae to avoid explicit boundary conditions, and the use of the "best" quadrature in the sense of Schoenberg. This quadrature is rarely used as it is difficult to calculate the quadrature coefficients to a sufficient precision. In this work we present a new method to calculate these coefficients, which solves this problem.

This paper is organised as follows. Section 2 presents the physical system being modelled. Section 3 presents the different schemes used to resolve the system. Section 4 presents the details of all numerical methods relying on splines including the quadrature method which is explained in Section 4.3. Section 5 discusses the convergence of the different methods for non-equidistant points. Section 6 presents the non-equidistant simulations and discusses the gains that were achieved as compared to equidistant points. Finally section 7 gives our conclusions.

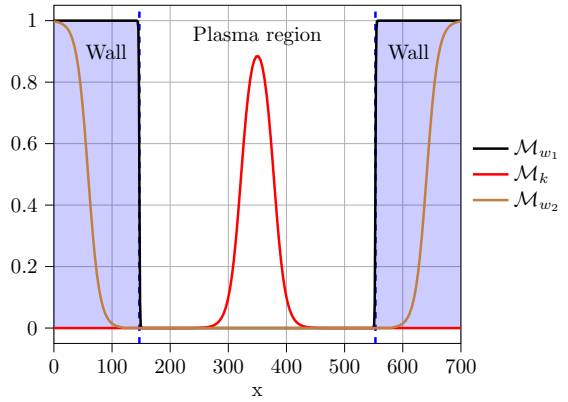
## 2 Physical System

The plasma sheath is the small layer adjacent to a wall in a plasma. In this region a positive charge develops, thus creating an electric potential which accelerates ions towards the wall, and confines electrons. This ensures that the plasma can attain a steady state with an appropriate source to balance the losses through the sheath. The plasma then remains quasi-neutral at typical scales longer than the Debye length.

The sheath model used for the simulations in this paper describes a highly simplified situation. The domain is considered to be one-dimensional. This dimension describes a line which follows




(a) A poloidal cut of a tokamak. The simulation follows a line inside the Scrape-Off Layer (SOL) shown in yellow from one divertor to the other. This line is outside the Last Closed Flux Surface (LCFS).




(b) The simulation domain following a magnetic field line in the SOL. Mask functions indicate whether a given point is in the wall. The wall region is shown in blue. $\mathcal{M}_{w_1}$ is a mask describing the absorption of particles by the wall, $\mathcal{M}_{w_2}$ is a mask describing the absorption of momentum and energy by the wall, and $M_k$ is a mask describing the kinetic source.

Figure 1: The 1D spatial simulation domain.

a magnetic field line, beginning and ending in a wall. This situation can be seen in figure 1 (see figure 1a for a 2D diagram of a tokamak plasma cross-section, and figure 1b for the 1D simulation domain). The simulation consists of three regions, two wall regions and the plasma region.

At the edge of the vessel the plasma interacts with the wall. Electrons move faster than ions, and would be rapidly absorbed by the wall. To prevent an overly large charge difference building in this area, violating quasi-neutrality, electron losses must be inhibited. As a result a potential difference is created which accelerates the ions towards the wall, and decelerates electrons approaching the wall, restoring conditions for a weak plasma current loss.

As a result of this behaviour the electric potential has a very steep gradient close to the wall [1]. This can be understood as a standing shock wave localised at the wall. In order to take into account the behaviour due to this gradient it is important to resolve the system with a small spatial resolution in this area. This prevents the wall being seen as a discontinuity in the distribution function. Discontinuities can propagate through a simulation causing non-physical behaviour such as oscillations.

This high resolution results in costly simulations with little gain outside of the sheath area. Non-equidistant points are a valuable tool for circumventing this problem. However many of the numerical methods commonly used in such simulations are optimised for equidistant points.

## 2.1 Vlasov-Poisson

The system is known as the Vlasov-Poisson system as it is composed of a Vlasov equation and a Poisson equation. The Vlasov equation (1) is an advection equation describing the evolution of the distribution function $f_s(t,x,v)$ for electrons ($s=e$), and ions ($s=i$). The Poisson equation (2) describes the quasi-neutrality condition:

$$\partial_t f_s(t,x,v) + v\partial_x f_s(t,x,v) - \frac{q_s}{m_s}\partial_x\phi(t,x)\partial_v f_s(t,x,v) = S_s(t,x,v) + C_{ss}(t,x,v) \tag{1}$$

$$\partial_x^2\phi(t,x) = -\frac{\rho_q(t,x)}{\varepsilon_0} \tag{2}$$

Where $x$ is the spatial coordinate, $v$ is the velocity coordinate, $t$ is the temporal coordinate, $q_s$ and $m_s$ are respectively the charge and mass of the species $s$, $S_s(t,x,v)$ is a source term, $C_{ss}(t,x,v)$ is a intra-species collision operator, $\phi(t,x)$ is the electric potential, $\rho_q$ is the charge density, and $\varepsilon_0$ is the permittivity of free space.

The charge density $\rho_q$ is defined from the particle density $n_s$ as follows:

$$\rho_q(t,x) = \sum_s q_s n_s(t,x) \tag{3}$$

$$n_s(t,x) = \int f_s(t,x,v)dv \tag{4}$$

The collision operator $C_{ss}(t,x,v)$ resembles the operator used in GYSELA [8], and is defined as follows:

$$C_{ss}(t,x,v) = \partial_v \left[ D_v(t,x,y(x,v))\partial_v f_s(t,x,v) + f_s(t,x,v) D_v(t,x,y(x,v)) m_s \frac{v - V_M(t,x)}{T_M(t,x)} \right] \tag{5}$$

where $V_M(t,x)$ and $T_M(t,x)$ are functions which ensure that the collision operator correctly conserves the momentum and energy, and $D_v$ is a diffusion coefficient defined as:

$$D_v(t,x,y(x,v)) = D_0(t,x) \left[ \frac{\psi(y)}{y} - \frac{\psi(y)}{2y^3} + \frac{\psi'(y)}{y^2} \right] \tag{6}$$

$$D_0(t,x) = \frac{3\sqrt{2\pi}T_s(t,x)}{4m_s}\nu_{ss} \tag{7}$$

$$T_s(t,x) = \int \frac{m_s(v - V_s(t,x))^2}{n_s(t,x)} f_s(t,x,v)dv \tag{8}$$

$$V_s(t,x) = \int \frac{v}{n_s(t,x)} f_s(t,x,v)dv \tag{9}$$

$$\psi(y) = \frac{2}{\sqrt{\pi}} \int_0^y \exp\left(-z^2\right)\, dz \tag{10}$$

$$y(x,v) = \frac{|v|\sqrt{m_s}}{\sqrt{2T_s(t,x)}} \tag{11}$$

where $\nu_{ss}$ is the two-species collision frequency for collisions between particles of the same species $s$.

The constants $V_M(t,x)$ and $T_M(t,x)$ are the solutions to the following system of equations:

$$\langle D_v \rangle V_M + \frac{\langle D_v' \rangle}{m_s} T_M = \langle v D_v \rangle \tag{12}$$

$$\langle v D_v \rangle V_M + \frac{\langle (v D_v)' \rangle}{m_s} T_M = \langle v^2 D_v \rangle \tag{13}$$

The operator $\langle \cdot \rangle$ is defined as:

$$\langle G \rangle = \int_{\mathbb{R}} G f_s(t,x,v)dv \tag{14}$$

The source term $S_s(t,x,v)$ contains three different terms:

$$S_s(t,x,v) = S_{s,w_1} + S_{s,w_2} + S_{s,k} \tag{15}$$

$S_{s,w_1}$ is a sink term describing the loss of particles in the wall. This operator conserves the charge. $S_{s,w_2}$ is an additional sink term describing the loss of momentum and energy in the wall. Both sink terms are Bhatnagar-Gross-Krook operators [9]. $S_{s,k}$ is a kinetic source term similar to the one used in GYSELA [10], which describes the addition of particles and energy from the body of the plasma in order to compensate for the particles lost into the wall. Once a steady state is reached, this operator ensures that the total number of particles remains constant. The terms are defined as follows:

$$S_{s,w_1}(t,x,v) = -\nu_{s,w_1}(t,x)\mathcal{M}_{w_1}(x)[f_s(t,x,v) - g_{s,w}(n_w, T_{w_1}, v)] \tag{16}$$

$$S_{s,w_2}(t,x,v) = -\nu_{w_2}\mathcal{M}_{w_2}(x)\left[f_s(t,x,v) - g_{s,w}(n_s(t,x), T_{w_2}(t), v)\right] \tag{17}$$

$$S_{s,k}(t,x,v) = \frac{\mathcal{M}_k(x)}{\int_0^{L_x} \mathcal{M}_k(x')dx'} \frac{s_k\sqrt{m_s}}{\sqrt{2\pi T_k}} e^{-\frac{m_s v^2}{2T_k}} \tag{18}$$

where $\nu_{s,w_1}(t,x)$ is the adjusted amplitude of the particle sink for species $s$, $\mathcal{M}_z(x)$ are mask functions (with $z \in \{w_1, w_2, k\}$), $g_{s,w}(n,T,v)$ is the distribution function targeted by the sink operators in the wall region defined in equation (19), $n_w$ is the target density in the wall, $T_{w_1}$ is

the target temperature in the wall, $\nu_{w_2}$ is the constant amplitude for the momentum and energy sink in the wall, $n_s(t,x)$ is the particle density defined in equation (4), $T_{w_2}(t)$ is the target temperature defined in equation (20), $s_k$ is the magnitude of the source term and $T_k$ is the temperature of the source.

The distribution function $g_{s,w}(n,T,v)$, and target temperature for the momentum and energy sink $T_{w_2}(t)$ are defined as follows:

$$g_{s,w}(n,T,v) = \frac{n\sqrt{m_s}}{\sqrt{2\pi T}} \exp\left(-\frac{m_s v^2}{2T}\right) \tag{19}$$

$$T_{w_2}(t) = \frac{1}{6}\left[\frac{\int[n_s(t,x_{Lw})v - \int v f(t,x_{Lw},v)dv]^2 f dv}{n_s(t,x_{Lw})^3} + \frac{\int[n_s(t,x_{Rw})v - \int v f(t,x_{Rw},v)dv]^2 f dv}{n_s(t,x_{Rw})^3}\right] \tag{20}$$

where $x_{Lw}$ and $x_{Rw}$ are the left and right boundaries between the plasma region and the wall region.

The adjusted amplitudes of the particle sink $\nu_{s,w_1}(t,x)$ are chosen such that the operator conserves charge locally. They are defined such that they respect the following equation:

$$\nu_{iw_1}(t,x) = \nu_{w_1} \tag{21a}$$

$$\nu_{ew_1}(t,x) = \nu_{w_1}\frac{n_i(t,x) - n_w}{n_e(t,x) - n_w} \tag{21b}$$

where $\nu_{w_1}$ is the constant amplitude for the particle sink in the wall. This makes the operator somewhat unusual for a BGK operator, in that the coefficients are not only space-dependant, but also time-dependant. This specificity means that the equation cannot be solved analytically, instead a Runge-Kutta scheme will be used.

The mask functions $\mathcal{M}_z(x)$ allow different behaviour to be specified in the wall and in a central region where a particle source is added. They are defined as follows:

$$\mathcal{M}(x,x_L,x_R,d) = \frac{1}{2}\left[\tanh\left(\frac{x-x_L}{d}\right) - \tanh\left(\frac{x-x_R}{d}\right)\right] \tag{22}$$

$$\mathcal{M}_w(x) = 1 - \mathcal{M}(x,x_{Lw},x_{Rw},d_w) \tag{23}$$

$$\mathcal{M}_k(x) = \mathcal{M}(x,x_{Lk},x_{Rk},d_k) \tag{24}$$

$$\mathcal{M}_{w_2}(x) = \left[1 - \mathcal{M}\left(x, \frac{x_0+4x_{Lw}}{5}, \frac{x_N+4x_{Rw}}{5}, \frac{x_{Lw}-x_0}{125}\right)\right] \cdot \left[1 - \mathcal{M}\left(x, \frac{3x_0+2x_{Lw}}{5}, \frac{3x_N+2x_{Rw}}{5}, \frac{x_0+4x_{Lw}}{30}\right)\right] \tag{25}$$

where $x_{Lk}$ and $x_{Rk}$ are the left and right boundaries of the region in which the plasma source is located, and $d_w$ and $d_k$ are input parameters which control the steepness of the masks at the transition between regions. In the real world $d_w$ would be infinitely small, however this cannot be resolved in our simulation without creating unphysical oscillations. The presence of this parameter therefore provides us with a transition region. It is important that there are sufficiently many points along this region to represent the transition without introducing unphysical behaviour. However it is equally important to strive to use as sharp a mask as possible in order to have a more realistic simulation.

The shape described by equations (23) - (25) can be seen in figure 1b. This use of a mask function is a penalisation technique, as described by Paredes et al. [11]. The same mask function has previously been used in a very similar problem by Caschera et al. [12].

The equations used in the simulation are unitless. This is achieved by normalising the variables in equations (1)-(13). Each unitless variable $\hat{V}$ is defined as follows:

$$\hat{V} = \frac{V}{V^\star} \tag{26}$$

where $V$ is $\hat{V}$ expressed in a given set of units, and $V^\star$ is the normalisation parameter, as defined in table 1.

| Variable | Symbol | Normalisation parameter | Definition |
|---|---|---|---|
| Density | $n_s$ | Reference density | $n_0$ |
| Temperature | $T_s$ | Reference temperature | $T_0$ |
| Mass | $m_s$ | Electron mass | $m_e$ |
| Charge | $q_s$ | Proton Charge | $e$ |
| Time | $t$ | Inverse of the electron plasma frequency | $\frac{1}{\omega_{pe}} = \sqrt{\frac{m_e \varepsilon_0}{n_0 e^2}}$ |
| Length | $x$ | Debye length | $\lambda_{D_0} = \sqrt{\frac{\varepsilon_0 T_0}{n_0 e^2}}$ |
| Velocity of species $s$ | $v_s$ | Thermal velocity of species $s$ | $v_{ths} = \lambda_{D_0}\omega_{ps} = \sqrt{\frac{T_0}{m_s}}$ |
| Distribution function | $f_s$ | Reference density in phase space | $\frac{n_0}{v_{ths}}$ |
| Electric potential | $\phi$ | Reference electric potential | $\frac{T_0}{e}$ |
| Electric field | $E$ | Reference electric field | $\frac{T_0}{e\lambda_{D_0}}$ |
| Collision frequency | $\nu_{s,y}$ | Electron plasma frequency | $\omega_{pe}$ |

Table 1: The parameters used to normalise the different variables that appear in the equations. The index $_s$ denotes the species (ion or electron). The index $_y$ denotes the region (wall or plasma). $\varepsilon_0$ is the permittivity of free space.

Equations (1) and (2) are therefore expressed as:

$$\left[\partial_{\hat{t}} + \frac{1}{\sqrt{\hat{m}_s}}\left(\hat{v}_s\partial_{\hat{x}} - \hat{q}_s\partial_{\hat{x}}\hat{\phi}(\hat{t},\hat{x})\partial_{\hat{v}_s}\right)\right]\hat{f}_s(\hat{t},\hat{x},\hat{v}_s) = \hat{S}_s(\hat{t},\hat{x},\hat{v}_s) + \hat{C}_s(\hat{t},\hat{x},\hat{v}_s) \tag{27}$$

$$\partial^2_{\hat{x}}\hat{\phi}(\hat{t},\hat{x}) = -\hat{\rho}_q(\hat{t},\hat{x}) \tag{28}$$

Where $\hat{m}_s$ is the normalised mass of species $s$, $\hat{v}_s$ is the velocity normalised for species $s$, $\hat{x}$ is the normalised spatial coordinate, $\hat{q}_s$ is the normalised charge for species $s$, $\hat{\phi}(\hat{t},\hat{x})$ is the normalised electric potential, $\hat{S}_s(\hat{t},\hat{x},\hat{v}_s)$ is the normalised source term, $\hat{C}_s(\hat{t},\hat{x},\hat{v}_s)$ is the normalised collision term, and $\hat{\rho}_q(\hat{t},\hat{x})$ is the charge density.

For simplicity the notation $\hat{\cdot}$ will be dropped in the rest of the paper.

## 2.2 Conservation Laws

The equations described above are kinetic equations. By multiplying equation (27) by powers of $v_s$ and integrating, conservation equations can be obtained from the kinetic equations. This gives a fluid model of our simulation which depends on the fluid density $n_s$ (defined in equation (4)), the particle flux $\Gamma_s$, the Reynold's stress $\Pi_s$ and the heat flux $Q_s$. These quantities are defined as follows:

$$\Gamma_s(t,x) = \int v_s f_s(t,x,v_s)dv_s \tag{29}$$

$$\Pi_s(t,x) = \int v_s^2 f_s(t,x,v_s)dv_s \tag{30}$$

$$Q_s(t,x) = \int \frac{1}{2}v_s^3 f_s(t,x,v_s)dv_s \tag{31}$$

The conservation equations describe the conservation of the particle density (32), the mean velocity (33), and the kinetic energy (34):

$$\partial_t n_s(t,x) + \frac{1}{\sqrt{m_s}}\partial_x\Gamma_s(t,x) = \int S_s(t,x,v_s) + C_{ss}(t,x,v_s)dv_s \tag{32}$$

$$\partial_t\Gamma_s(t,x) + \frac{1}{\sqrt{m_s}}\left(\partial_x\Pi_s(t,x) + q_s\partial_x\phi(t,x)n_s(t,x)\right) = \int v_s S_s(t,x,v_s) + v_s C_{ss}(t,x,v_s)dv_s \tag{33}$$

$$\partial_t \Pi_s(t,x) + 2\frac{1}{\sqrt{m_s}}\left(\partial_x Q_s(t,x) + q_s \partial_x \phi(t,x)\Gamma_s\right) = \int v_s^2 S_s(t,x,v_s) + v_s^2 C_{ss}(t,x,v_s) dv_s \tag{34}$$

These equations therefore provide a method for evaluating the error of the system. This is done by comparing numerical approximations of the left and right-hand sides of equations (32)-(34). These errors will be used in Section 6 to evaluate the precision of the numerical schemes.

# 3 Semi-Lagrangian Scheme

In this section the main methods used to resolve the system defined by equations (27) and (28) will be described. Namely the time stepping method, the source and sink operators, the collision operator, the semi-Lagrangian advection, and the Poisson solver. Three of these methods are based on splines. The splines themselves will be discussed in detail in Section 4.

The numerical schemes used in the simulation are all accurate to at least third order in space and second order in time.

The data will be stored on a non-periodic grid representing the domain $[x_0, x_{N_x-1}] \times [v_0, v_{N_v-1}]$, where $N_x$ and $N_v$ are respectively the number of interpolation points in the spatial and velocity dimensions. One particularity of this paper is that non-equidistant points are used in both dimensions.

## 3.1 Time Stepping

The system described by equations (27) and (28) is solved using a predictor-corrector scheme. Strang's second order splitting scheme [13] is used to calculate $\partial_t f_{s,n} = \partial_t f_s(t_n, x, v_s)$ at each step of the scheme. Equation (27) is therefore broken up into six equations: a spatial advection equation (35), a velocity advection equation (36), two equations describing the sinks (37),(38), an equation describing the source (39), and an equation describing the collisions (40):

$$\partial_t f_{s,n} = -\frac{1}{\sqrt{m_s}} v_s \partial_x f_{s,n} \tag{35}$$

$$\partial_t f_{s,n} = \frac{q_s}{\sqrt{m_s}} \partial_x \phi_n \partial_{v_s} f_{s,n} \tag{36}$$

$$\partial_t f_{s,n} = -\nu_{s,w_1}(t,x)\mathcal{M}_{w_1}(x)(f_{s,n} - g_{s,w}(n_w, T_{w_1}, v_s)) \tag{37}$$

$$\partial_t f_{s,n} = -\nu_{w_2}\mathcal{M}_d(x)(f_{s,n} - g_{s,d}(n_s(t,x), T_{w_2}(t), v)) \tag{38}$$

$$\partial_t f_{s,n} = \frac{\mathcal{M}_k(x)}{\int_{x_0}^{x_{N_x-1}} \mathcal{M}_k(x')dx'} \frac{s_k}{\sqrt{2\pi T_k}} e^{-\frac{v_s^2}{2T_k}} \tag{39}$$

$$\partial_t f_{s,n} = \partial_{v_s}\left(D_{v_s}(t,x,v_s)\,\partial_{v_s} f_{s,n} + D_{v_s}(t,x,v_s)\frac{(v_s - V_M(t,x))}{T_M(t,x)} f_{s,n}\right) \tag{40}$$

where $\phi_n$ is the approximation of the function $\phi(t_n, x)$.

Equation (36) is chosen as the central operator of the Strang splitting.

Let us define $f_n$ the vector $\{f_{i,n}, f_{e,n}\}$, and six operators : $\mathcal{X}_{\Delta t}$, $\mathcal{V}_{\Delta t}(\phi_n)$, $\mathcal{W}_{\Delta t}$, $\mathcal{D}_{\Delta t}$, $\mathcal{K}_{\Delta t}$, $\mathcal{C}_{\Delta t}$ which are defined as the operators which solve the six equations (35)-(40) for $f_{i,n}$, then $f_{e,n}$.

The operator $\mathcal{A}_{\Delta t}$ which solves equation (27) is therefore defined as:

$$\mathcal{A}_{\Delta t}(f_n, \phi_n) = \left(\mathcal{W}_{\frac{\Delta t}{2}}\mathcal{C}_{\frac{\Delta t}{2}}\mathcal{K}_{\frac{\Delta t}{2}}\mathcal{D}_{\frac{\Delta t}{2}}\mathcal{X}_{\frac{\Delta t}{2}}\mathcal{V}_{\Delta t}(\phi_n)\mathcal{X}_{\frac{\Delta t}{2}}\mathcal{D}_{\frac{\Delta t}{2}}\mathcal{K}_{\frac{\Delta t}{2}}\mathcal{C}_{\frac{\Delta t}{2}}\mathcal{W}_{\frac{\Delta t}{2}}\right) f_n \tag{41}$$

The second-order time stepping algorithm, which approximates the solution to the system described by equations (27) and (28), can finally be summarised as follows:

$$f_{n+1} = \mathcal{A}_{\Delta t}\left(f_n, \mathcal{P}\mathcal{A}_{\frac{\Delta t}{2}} f_n\right) \tag{42}$$

$$\phi_{n+1} = \mathcal{P}(f_{n+1}) \tag{43}$$

where $\mathcal{P}$ is the Poisson operator described in Section 3.3.

## 3.2 Vlasov

The advection equations (35) and (36) are solved using a backward semi-Lagrangian scheme. This scheme is a mixture between the well-known PIC and Eulerian methods [14]. The trajectories that particles located at each grid point would have followed, are traced back to find the location of the particles at the previous time step. These trajectories are known as characteristics and the location at the previous time step is known as the foot of the characteristic. During a simple advection, the value of the distribution function remains constant along a characteristic; therefore the value at the grid point is equal to the value at the foot of the characteristic. Figure 2 illustrates the location of the foot of the characteristic $x_6^\star$ for a grid point $x_6$. For constant advection the foot of the characteristic can be found trivially.

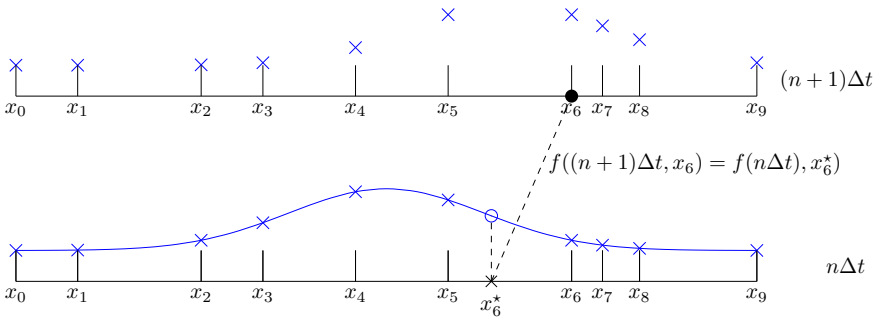



Figure 2: A visual description of the semi-Lagrangian scheme where $x_6^\star$ is the foot of the characteristic of $x_6$.

As seen in figure 2, the foot of the characteristic is not necessarily found at a grid point. Therefore an interpolation method is required to approximate the value at this point. In this work, a spline interpolation method is used. This is a common choice used in many codes [5, 7], including the GYSELA code [3] The values at the previous time step are used to construct a spline representation of the function. The spline representation can then be evaluated at any point on the domain. It is evaluated at the feet to provide the updated value of the function at the grid point. This is described in detail in Section 4.

The distribution function is assumed to be constant outside of the domain. Therefore if the foot of the characteristic falls outside the domain, the boundary value is used to update the value of the function at the grid point. This is justified by the fact that the distribution function falls to zero in the wall and tends to zero for large velocity magnitudes.

## 3.3 Poisson

The quasi-neutrality condition defined in equation (28) takes the form of a Poisson equation. It is solved using the finite element method (FEM) constructed on a spline basis. Most other codes, including the GYSELA code use alternative methods such as Fourier transforms [3] or finite differences (FD) [4, 5]. The choice was made to use FEM instead of FD as it is simpler to express the equations on non-uniform points. FEM is preferred over Fourier transforms as the latter place unnecessary restrictions on the geometry of the problem. As GYSELA will be extended in the future to include more complex geometries it is important to test methods which do not constrain these choices.

The spline basis used for the FEM is described in detail in Section 4. We use the notation $\{b_{0,d}(x), \ldots, b_{n_b-1,d}(x)\}$ to denote the spline basis for splines of degree $d$, where $n_b$ is the number of basis splines. As explained in Section 4, the chosen spline boundary conditions are such that $n_b$ is equal to the number of interpolation points ($N_x$ in the spatial direction, and $N_v$ in the velocity direction).

The approximations of the electric potential (44) and the charge density (45) are expressed on the spline basis as follows:

$$\tilde{\phi}_n(x) = \sum_{j=0}^{n_b-1} b_{j,d}(x)\, C_{\phi_j} \tag{44}$$

$$\tilde{\rho}_{n,q}(x) = \sum_{j=0}^{n_b-1} b_{j,d}(x)\, C_{\rho_{q,j}} \tag{45}$$

where $\tilde{\rho}_{n,q}(x)$ is the approximation of $\rho_q(t_n, x)$, and $C_{\phi_j}$ and $C_{\rho_{q,j}}$ are respectively the spline coefficients of the approximations $\tilde{\phi}_n(x)$ and $\tilde{\rho}_{n,q}(x)$.

This leads to the following expression for the weak form of equation (28):

$$\sum_{j=0}^{n_b-1} \left[ \partial_x b_{i,d}(x) b_{j,d}(x) \big|_{x_0}^{x_{N_x-1}} - \int_{x_0}^{x_{N_x-1}} \partial_x b_{i,d}(x) \partial_x b_{j,d}(x)\, dx \right] DC_{\phi_j} = \sum_{l=0}^{n_b-1} C_{\rho_{q,l}} \int_{x_0}^{x_{N_x-1}} b_{i,d}(x) b_{l,d}(x)\, dx \tag{46}$$

for all integer values $0 \leq i < n_b$.

Dirichlet boundary conditions are imposed on the electric potential:

$$\sum_{j=0}^{n_b-1} b_{j,d}(x_0)\phi_j = \sum_{j=0}^{n_b-1} b_{j,d}(x_{N_x-1})\phi_j = 0 \tag{47}$$

As explained in Section 4, the non-uniform spline basis is defined such that the subset $\{b_1^d, \ldots, b_{n_b-2}^d\}$ is a basis for splines respecting the Dirichlet condition. Equation (46) can therefore be simplified to:

$$-\sum_{j=1}^{n_b-2} \int_{x_0}^{x_{N_x-1}} \partial_x b_{i,d}(x) \partial_x b_{j,d}(x)\, dx\; C_{\phi_j} = \sum_{l=0}^{n_b-1} C_{\rho_{q,l}} \int_{x_0}^{x_{N_x-1}} b_{i,d}(x) b_{l,d}(x)\, dx \tag{48}$$

for all integer values $0 < i < n_b - 1$.

By allowing the index $i$ to vary over all functions in the reduced basis, equation (48) leads to a matrix equation of the form $S\vec{\phi} = \vec{r}$, where each of the elements of the matrix $S$ and the vector $\vec{r}$ are defined as:

$$S_{ij} = -\int_{x_0}^{x_{N_x-1}} \partial_x b_{i,d}(x) \partial_x b_{j,d}(x)\, dx \tag{49}$$

$$r_i = \sum_{l=0}^{n_b-1} \rho_{q,l} \int_{x_0}^{x_{N_x-1}} b_{i,d}(x) b_{l,d}(x)\, dx \tag{50}$$

The matrix $S$ is referred to as the stiffness matrix. The vector $\vec{\phi}$ contains the spline coefficients $\phi_j$.

There are several possible ways of calculating the integrals in equation (48). Different options including Gauss-Legendre quadrature, integration by parts, and recursive solutions using the properties of B-Splines were explored by Vermeulen et al. [15]. In this work we choose Gauss-Legendre quadrature as the sample points can be chosen such that the product of two splines of degree $d$ is integrated exactly. This is achieved by using $d+1$ sample points per cell. Thus no additional error is introduced unnecessarily.

Using equation (3), the charge density $\rho_{q,n}$ is defined as:

$$\rho_{q,n}(t,x) = \int f_{i,n}(t,x,v_i) dv_i - \int f_{n,e}(t,x,v_e)\, dv_e \tag{51}$$

This integral does not describe a product of splines so the options suggested by Vermeulen et al. [15] are not adapted to this situation. It would be possible to use these methods by first approximating the functions with a spline, however the calculation of the spline coefficients is a costly operation. To avoid this unnecessary cost, a quadrature method is used to calculate the charge density and all other integrals which don't involve splines. The chosen quadrature method is described in Section 4.3 and is derived from spline integration.

### 3.4 BGK Wall

The wall is described by equations (37) and (38), which are Bhatnar-Gross-Krook operators. In the case of equation (37), the coefficients $\nu_{s,w_1}$, defined in equation (21), are time-dependent, while in the case of equation (38), it is the function $g_{s,w}(n_s(t,x), T_d(t), v)$ which is time-dependent. These

are non-standard versions of the operator which require special handling, as it means that it is not possible to solve equations (37) and (38) exactly. Instead, a second-order explicit Runge-Kutta scheme, known as the midpoint method is used. This method is defined as follows:

$$f_{s,n+\frac{1}{2}} = f_{s,n} + \frac{\Delta t}{2}\partial_t f_{s,n} \tag{52a}$$

$$f_{s,n+1} = f_{s,n} + \Delta t \partial_t f_{s,n+\frac{1}{2}} \tag{52b}$$

where $f_{s,n} = f_s(t_n, \cdot, \cdot)$ is the approximation of the distribution function at time $t_n$ , $\Delta t$ is the time step, and $\partial_t f_{s,n}$ is given by equation (37) or (38).

Let us define the operator $\nu\left(f_n\right)$ which calculates the normalised frequencies given the approximations of the distribution functions at a given time $t_n$ using equations (4) and (21):

$$\nu\left(f_n\right) = \begin{cases} \nu_{i,w_1} = \nu_{w_1} \\ \nu_{e,w_1} = \nu_{w_1} \frac{\int f_{i,n}(x,v_i)dv_i - n_w}{\int f_{e,n}(x,v_e)dv_e - n_w} \end{cases} \tag{53}$$

The integrals required for this calculation are the same as those in equation (51) and will therefore be solved in the same way; namely using the spline quadrature scheme described in Section 4.3.

Once the normalised frequencies have been calculated, equations (37) and (38) can be solved analytically.

Let us define the operator $\mathcal{W}^*_{\Delta t}\left(f_n, \nu_s\right)$ which resolves equation (37) for known frequencies. The second-order operator $\mathcal{W}_{\Delta t}\left(f_n\right)$ is therefore defined as:

$$\mathcal{W}_{\Delta t}\left(f_n\right) = \mathcal{W}^*_{\Delta t}\left(f_n, \nu\left\{\mathcal{W}^*_{\frac{\Delta t}{2}}\left(f_n, \nu\left\{f_n\right\}\right)\right\}\right) \tag{54}$$

Similarly, by defining the operator $\mathcal{D}^*_{\Delta t}\left(f_n, g_{s,d}\right)$ which resolves equation (38) for a given density and temperature, we obtain:

$$\mathcal{D}_{\Delta t}\left(f_n\right) = \mathcal{D}^*_{\Delta t}\left(f_n, g\left\{\mathcal{D}^*_{\frac{\Delta t}{2}}\left(f_n, g\left\{f_n\right\}\right)\right\}\right) \tag{55}$$

### 3.5 Kinetic Source

The particle source evolution is defined in equation (39). This equation is sufficiently simple to allow it to be solved analytically. None of the terms depend on time or on the distribution function. Therefore the operator $\mathcal{K}_{\Delta t}$ is defined simply as:

$$\mathcal{K}_{\Delta t}(f_{s,n}) = f_{s,n} + \Delta t \frac{\mathcal{M}_k(x)}{\int_{x_0}^{x_{N_x-1}} \mathcal{M}_k(x')dx'} \frac{s_k}{\sqrt{2\pi T_k}} e^{-\frac{v_s^2}{2T_k}} \tag{56}$$

### 3.6 Collisions

The intra-species collisions are described by equation (40). By expressing the distribution function on the spline basis, this equation can be rewritten as:

$$\begin{aligned} \partial_t f_{s,n} = \sum_{i=0} \Bigg[ & \partial_v D_v(x_j, v)\, \partial_v b_{i,d}(v) + D_v(x_j, v)\, \partial_v^2 b_{i,d}(v) + \partial_v D_v(x_j, v) \frac{m_a(v - V_M(x_j))}{T_M(x_j)} b_{i,d}(v) \\ & + D_v(x_j, v) \frac{m_a}{T_M(x_j)} b_{i,d}(v) + D_v(x_j, v) \frac{m_a(v - V_M(x_j))}{T_M(x_j)} \partial_v b_{i,d}(v) \Bigg] C_{f^n_{i,j}} \end{aligned} \tag{57}$$

where $C_{f^n_{i,j}}$ are the coefficients of the splines representing the distribution function $f_{s,n}(x_j, v)$ along the velocity dimension, and $b_{i,d}$ are the basis splines of degree d.

This is a stiff equation so a semi-implicit method is used, namely the Crank-Nicolson scheme. In order to solve the equation implicitly it is assumed that $V_M(x)$, $T_M(x)$ and $D_v(x)$ vary sufficiently slowly to allow the following approximation:

$$V_M((n+1)\Delta t, x_j, v) \approx V_M(n\Delta t, x_j, v) \tag{58}$$

$$T_M((n+1)\Delta t, x_j, v) \approx T_M(n\Delta t, x_j, v) \tag{59}$$

$$D_v((n+1)\Delta t, x_j, v) \approx D_v(n\Delta t, x_j, v) \tag{60}$$

The derivative $\partial_v D_v(x,v)$ can be calculated analytically using the definition of $D_v(x,v)$. The integrals necessary to define the temperature $T_s(t,x)$ and average velocity $V_s(t,x)$ are calculated similarly to the density $n_s(t,x)$ using the spline quadrature described in Section 4.3.

The semi-implicit Crank-Nicolson scheme is expressed as follows:

$$\frac{f_s\left((n+1)\Delta t, x_j, v\right) - f_s\left(n\Delta t, x_j, v\right)}{\Delta t} = \frac{1}{2}\left[\mathcal{C}_{ss}((n+1)\Delta t, x_j, v) + \mathcal{C}_{ss}(n\Delta t, x_j, v)\right] \tag{61}$$

Combining equations (61) and (57) allows us to express the scheme as a matrix equation of the form:

$$(I - M)C_{f_j^{n+1}} = (I + M)C_{f_j^n} \tag{62}$$

where $C_{f_j^n}$ is a vector containing the coefficients of the spline representation of the distribution function at time $t = n\Delta t$, and at point $x_j$, and the matrix $M$ is defined as:

$$\begin{aligned} M_{li} = &\frac{\Delta t}{2}\Bigg\{ \left[\partial_v D_v(x_j, v_l)(v_l - V_M(x_j)) + D_v(x_j, v_l)\right] \frac{m_a}{T_M(x_j)} b_{i,d}(v_l) \\ &+ \left(\partial_v D_v(x_j, v_l) + D_v(x_j, v_l)\frac{m_a(v_l - V_M(x_j))}{T_M(x_j)}\right)\partial_v b_{i,d}(v_l) + D_v(x_j, v_l)\,\partial_v^2 b_{i,d}(v_l)\Bigg\} \end{aligned} \tag{63}$$

### 3.7 Conservation

The error in the conservation equations (32) - (34) is calculated by approximating each of the terms using the calculated distribution function and comparing the left and right-hand side of the equations.

All derivatives in these equations are calculated using spline interpolation. All integrals in these equations are calculated using the spline quadrature described in Section 4.3.

Numerical approximations of the $L_2$ and $L_\infty$ norms of the errors will be studied. The $L_2$ norm is calculated using the trapezoid rule.

Equations (29) - (34) contain integrals which are calculated on the domain $[-\infty, \infty]$, however a numerical representation cannot capture the entire domain. This means that the choice of grid can influence the conservation error. A larger domain in the velocity dimension will lead to less potential error in the conservations. Despite this, it is not necessary to have a very large domain in the velocity dimension. The distribution function can be approximated by a Maxwellian function in the velocity direction. Maxwellians can be truncated at relatively low values without causing large errors in the integrals. This is detailed in A. For our simulations we will use the domain $[-6, 6]$ which would lead to an error of $4.95 \cdot 10^{-9}$ for a Maxwellian function centred on 0.

## 4 Splines

As detailed in Section 3, splines underpin several of the methods used in the chosen numerical schemes. It is therefore important to fully understand their usage and the differences in their implementation for equidistant or non-equidistant schemes. There are many existing works which provide a detailed description of splines [16, 17, 18, 19], here we will give a brief description to clarify the notation used.

Splines are piecewise polynomial functions. A spline of degree $d$, defined over a domain $[a, b]$ is defined via a knot vector $[k_{-d}, ..., k_{n_c+d}]$ which we will consider to be such that:

$$k_{-d} \leq k_{1-d} \leq \cdots \leq k_0 = a < k_1 < \cdots < k_{n_c-1} < k_{n_c} = b \leq \ldots k_{n_c+d-1} \leq k_{n_c+d}$$

This choice affects the shape of the b-splines. A figure showing the resulting shape can be seen in appendix B.

The $n_c$ polynomial sections of the splines are each defined on a domain $[k_i, k_{i+1}]$, with $0 \leq i < n_c$.

Each spline is characterised by $n_b = n_c + d$ coefficients and basis functions $b_{i,d}(x)$:

$$\begin{aligned} b_{i,0}(x) &= \begin{cases} 1 & \text{if } k_i \leq x \leq k_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ b_{i,d}(x) &= \delta_{k_{i+d}-k_i\neq 0}\frac{x - k_i}{k_{i+d} - k_i} b_{i,d-1}(x) + \delta_{k_{i+d+1}-k_{i+1}\neq 0}\frac{k_{i+d+1} - x}{k_{i+d+1} - k_{i+1}} b_{i+1,d-1}(x) \end{aligned} \tag{64}$$

with:

$$\delta_c = \begin{cases} 1 \text{ if c} \\ 0 \text{ otherwise} \end{cases} \tag{65}$$

There are multiple possible definitions of the b-splines. The splines above are obtained by choosing to enforce partition of unity:

$$\sum_i b_{i,d}(x) = 1 \quad \forall x \in [a, b] \tag{66}$$

A spline $S_d(x)$ is represented on its basis as follows:

$$S_d(x) = \sum_{i=0}^{n_b-1} c_i b_{i,d}(x) \tag{67}$$

where the coefficients $c_i$ are the coefficients of the splines.

In order to use splines there are therefore two crucial steps. The first is the calculation of the coefficients of the splines given a function to be interpolated. The second is the evaluation of the spline at a given point.

## 4.1 Coefficient Calculation

The calculation of the coefficients of the spline which interpolates a given function $g(x)$ can be found by solving a matrix equation of the form:

$$B\vec{c} = \vec{g} \tag{68}$$

where the matrix $B \in \mathbb{R}^{n_b \times n_b}$ is an interpolation matrix, $\vec{c}$ is a vector containing the coefficients of the spline and $\vec{g}$ is a vector containing the value of the function at the interpolation points $\{y_i\}$. The i-th element of the vector $\vec{g}$ is defined as:

$$g_i = g(y_i) \tag{69}$$

The choice of interpolation points is important for the definition of the interpolation matrix. In this work we will use Greville abcissae as interpolation points. This leads to different points than the ones used in the GYSELA code [3], however they have two advantages. Firstly they ensure that the interpolant is bounded independently of the choice of knots [20]. Secondly, with strategic placement of the knots, they can remove the need for an explicit boundary condition. This could potentially be very useful in GYSELA where the value at the boundaries is not usually known.

Greville abcissae are defined as follows [19]:

$$y_i = \frac{\sum_{j=1}^{d} k_{i+j}}{d} \tag{70}$$

By choosing the boundary knots such that $k_{-d} = k_{1-d} = \cdots = k_0$ and $k_{n_c} = k_{n_c+1} = \cdots = k_{n_c+d}$, all the Greville abcissae are within the domain of interest so no additional boundary conditions are required for the definition of the matrix. This unusual choice can be particularly useful in cases where the boundary conditions are unknown, as is the case in the GYSELA code [3].

The interpolation matrix and the vector $\vec{g}$ are defined as follows:

$$B_{ij} = b_{j,d}(y_i) \tag{71}$$

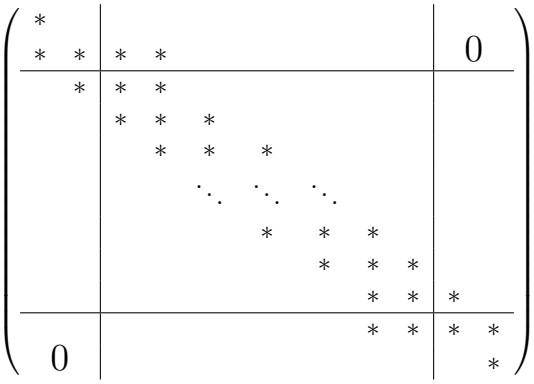
$$g_i = g(y_i) \tag{72}$$

This choice leads to a non-equidistant set of interpolation points, which must be taken into account in the parts of the code which are not based on splines. This is the case even if uniform knots are used, and means that the user does not have complete control over the choice of the points on which the simulation evolves. This choice differentiates the paper from many others which use the knots as interpolation points [3].

In the optimised case of equidistant knots, we cannot choose the boundary knots to be such that $k_{-d} = k_{1-d} = \cdots = k_0$ and $k_{n_c} = k_{n_c+1} = \cdots = k_{n_c+d}$ as they would no longer be equidistant. As a result some Greville points are found outside the domain. In this case boundary conditions are required. In order to easily compare the equidistant and non-equidistant case the boundary

conditions are provided by adding additional interpolation points inside the domain. These points are placed such that the interpolation points coincide for the equidistant and non-equidistant schemes.

The matrix $B$ is a positive-definite banded matrix [21] with $d$ upper and lower diagonals. Equation (68) can easily be solved numerically with a banded solver, however this solution can be costly. With our chosen interpolation points, in the centre of the matrix the band is narrower with $k \leq d-1$ upper and lower diagonals. In fact, in the equidistant case, thanks to symmetry properties, the band is even narrower. It has only $k = \left\lfloor \frac{d}{2} \right\rfloor$ upper and lower diagonals, where $\lfloor \cdot \rfloor$ is the floor operator. The sparsity pattern resembles the following (example given for d=3):



In this work an optimisation is made to the solver, the matrix is pivoted such that the first $l$ lines, corresponding to the boundary region, are moved to the bottom right corner of the matrix. The resulting matrix has a block format and can be inverted using the Schur's complement and a banded solver. This manipulation increases the speed of the calculation and reduces the necessary storage requirements as the banded block can be narrower.

## 4.2 Evaluation

Evaluating a spline at a given point $x$ is a simple calculation once the spline coefficients are known:

$$S_d(x) = \sum_i c_i b_{i,d}(x) \tag{73}$$

This calculation can then be reduced further by considering the support of the basis functions. We therefore have:

$$S_d(x) = \sum_{i=i^*}^{i^*+d} c_i b_{i,d}(x) \tag{74}$$

where $i^*$ is the index of the last knot before $x$.

In the case of uniform splines the knots $\{k_i\}$ are defined as $k_i = a + ih$ with $h = \frac{(b-a)}{n_c}$. It is therefore trivial to find $i^*$:

$$i^* = \left\lfloor \frac{x-a}{h} \right\rfloor \tag{75}$$

In the non-uniform case the knots $\{k_i\}$ inside the domain are placed arbitrarily, while the knots outside the domain are chosen such that:

$$k_{-d} = k_{1-d} = \cdots = k_0 < k_1 < \cdots < k_{n_c-1} < k_{n_c} = \ldots k_{n_c+d-1} = k_{n_c+d}$$

$i^*$ can therefore no longer be found trivially; instead a binary search algorithm is used. The average and worst cases for this algorithm are $\mathcal{O}(\log n_c)$ [22].

When solving an advection equation with the semi-Lagrangian method as described in Section 3.2 an array of $n_b \approx n_c$ values must be evaluated. This can therefore be quite costly as the algorithm is now $\mathcal{O}(n_c \log n_c)$. The cost can be reduced in one of two ways. Either parallelisation can allow us to calculate $i^*$ in parallel, or in a serial program we could benefit from the fact that the advection is constant and therefore the array of values is sorted. $i^*$ could therefore be found using a basic linear search which uses the $i^*$ associated with the previous point as a starting point for the search.

The remaining complexity in the evaluation of the spline comes from the calculation of the $d+1$ values $b_{i,d}(x)$ for $i^* \leq i \leq i^* + d$. The efficiency of this calculation depends on the limitations that are placed on the system. The most efficient calculation is available in the most restrictive case, where the degree is fixed and the points are equidistant. In the case of cubic splines, 19 FLOPs are required for the calculation of $b_{i,d}(x)$. In this work we use algorithms for which the degree is a parameter. Assuming equidistant points, $\{b_{i,d}(x)\}$ can therefore be calculated with $9d(d+1)/2+5$ FLOPs (59 FLOPs for cubic splines)[1]. If the knots are non-equidistant then $\{b_{i,d}(x)\}$ can be calculated with $d(4d+6)$ FLOPs (54 FLOPs for cubic splines). It is interesting to note that the non-equidistant case uses fewer FLOPs than the equidistant case, however it requires the storage of an additional $2d$ variables and the aforementioned binary search. The details of the calculation can be found in C.

## 4.3 Spline Quadrature

In this section the spline-based quadrature scheme is presented. Splines are easy to integrate. This means that a numerical integration method can be obtained via the spline representation of a function. The calculation of the spline representation is a relatively costly operation, however this step can be precalculated and integrated into quadrature coefficients.

In Section 4.3.1 we explain how quadrature coefficients are obtained from a spline representation. In Section 4.3.2 we discuss different methods for calculating the quadrature coefficients as precisely as possible. A new method is proposed which ensures the quadrature coefficients can be calculated precisely even for large degrees and many points. This method is described in detail in Section 4.3.3. Finally the precision of the different methods is compared in Section 4.3.4.

### 4.3.1 Quadrature coefficients from spline representations

Let us note $S_d(x)$ the spline representation of a function $g(x)$ with degree $d$. This spline and its integral can be expressed in terms of the spline coefficients $c_i$, and b-splines $b_{i,d}(x)$:

$$g(x) \approx S_d(x) = \sum_i c_i b_{i,d}(x) \tag{76}$$

$$\int S_d(x)dx = \sum_i c_i \int b_{i,d}(x)dx \tag{77}$$

Let us define a vector $\vec{b}_d$ such that the i-th element $(b_d)_i = \int b_{i,d}(x)dx$, and a vector $\vec{c}$ such that the i-th element is the i-th spline coefficient $c_i$. We can therefore express the above integration scheme as a scalar product:

$$\int S_d(x)dx = \vec{c} \cdot \vec{b}_d \tag{78}$$

As described in Section 4.1, the coefficients $\vec{c}$ are determined by solving the following matrix equation:

$$B\vec{c} = \vec{g} \tag{79}$$

where $B$ is an interpolation matrix, and $\vec{g}$ is the vector:

$$g_i = \begin{cases} g(y_i) & \text{if } d < i \leq n_c + d + 1 \\ 0 & \text{otherwise} \end{cases} \tag{80}$$

The quadrature is therefore expressed as:

$$\int S_d(x)dx = \left(B^{-1}\vec{g}\right) \cdot \vec{b}_d = \left(\vec{b}_d\right)^T B^{-1}\vec{g} = \vec{g}^T \left(B^{-1}\right)^T \vec{b}_d = \vec{g}^T \left(B^T\right)^{-1} \vec{b}_d \tag{81}$$

In other words, the quadrature coefficients $q_i$ can be obtained by solving the equation:

$$B^T \vec{q}_* = \vec{b}_d \tag{82}$$

[1] A division is assumed to be 4 FLOPs.

and taking $q_i = \vec{q}_{*,i+d}$. The first and last $d$ elements of the vector $\vec{q}_*$ can be discarded as, to obtain the approximation of the integral, they are multiplied by the first and last $d$ elements of the vector $\vec{g}$, which are all zero. The remaining $n_c + 1$ elements provide the same number of quadrature coefficients as there are quadrature points to fully define the quadrature scheme. The coefficients $q_i$ can be precalculated at the start of the simulation to avoid the unnecessary cost of solving a spline interpolation at every integration equation.

The integrals of the b-splines can be calculated trivially [23]:

$$\int b_{i,d}(x)dx = \frac{k_{i+d+1} - k_i}{d+1} \tag{83}$$

Thanks to our choice of knots the support of all the b-spline functions is inside the domain $[a, b]$ so equation (83) is sufficient to define the vector $\vec{b}_d$.

With this information it is simple to construct a quadrature scheme based on splines. However the scheme obtained using the interpolation matrix described in Section 4.1 is not optimal. Schoenberg has shown that the "best" quadrature (by which we mean the quadrature which minimises the error for a given degree $d$ and set of points $\{x_i\}$) is one derived by integrating a spline approximation of the function interpolated at the break points, with natural boundary conditions such that the spline is of degree $d$ at the edge of the domain [24].

Natural boundary conditions for a spline $S_{2d+1}(x)$ of degree $2d+1$ define the following $d+1$ equations:

$$\partial_x^r S_{2d+1}(k_0) = \partial_x^r S_{2d+1}(k_{N_c}) = 0 \quad \forall d < r \leq 2d \tag{84}$$

The "best" quadrature of degree $d$ is therefore the integral of a spline approximation of degree $2d+1$ with natural boundary conditions and interpolation points found at the break points.

### 4.3.2 Coefficient Calculation Methods

This "best" quadrature can be obtained using equation (82) and a different definition of the interpolation matrix describing natural boundary conditions. For this interpolation matrix, the first and last $d$ equations describe the boundary conditions while the central section remains similar to what was described in Section 4.1. The simplest way to describe the boundary conditions is using the fact that the derivatives of the spline are known. This gives us the following equations:

$$B_{i,j} = \partial_x^{(i+d+1)} b_{j,d}(k_0) \quad \forall 0 \leq i < d \tag{85}$$

$$B_{i+d,j} = b_{j,d}(k_i) \quad \forall d \leq i \leq N_c + d \tag{86}$$

$$B_{N_c+2d+1-i,j} = \partial_x^{(i+d+1)} b_{j,d}(k_{N_c}) \quad \forall 0 \leq i < d \tag{87}$$

This way of defining the boundary conditions is the traditional method used when calculating the coefficients of a spline with Hermite boundary conditions (providing the first $d$ derivatives at the boundary). However as we will see in table 2, this method is poorly conditioned due to the high order of both the spline and the derivatives.

An alternative method for determining the quadrature coefficients was proposed by Secrest [25]. However, this method is also poorly conditioned (see table 2). The conditioning of interpolation matrices tends to increase with the degree of the spline being interpolated. As a result this is a critical problem for this method where large degrees are required.

Although quadrature coefficients can be calculated in advance using arbitrary precision tools such as Mathematica, this is impractical when the position of the quadrature points is liable to change frequently depending on the choice of simulation. In this paper we additionally propose and use a new formulation which is significantly less susceptible to conditioning problems.

The conditioning of the matrices can also be improved somewhat through line normalisation. This can be done in the boundary region without worrying about the consequences for the right-hand side of the matrix equation as the right-hand side corresponding to these lines contains only zeros. Each line of the matrix is divided by its $L_1$ norm. For the central lines described by equation (86), the $L_1$ norm is equal to 1 as $b_{i,d} \geq 0$ and through the choice of the partition of unity normalisation as described by equation (66).

**Proposition 1.** *Let $A$ be a non-singular matrix. Let $M$ be a diagonal matrix such that the diagonal contains the $L_1$ norm of each line of the matrix $A$: $M_{i,i} = \sum_j |a_{ij}|$, let $\kappa(\star)$ denote the condition number on the $\infty$-norm, then*

$$\kappa\left(M^{-1}A\right) \leq \kappa\left(A\right)$$

*Proof.* The $\infty$-norm of $A$ is defined as:

$$\|A\|_\infty = \max_i \left\{ \sum_j |a_{ij}| \right\} = \max_i \{m_{i,i}\}$$

$M$ is defined such that $\left\|M^{-1}A\right\|_\infty = 1$.

$$\kappa\left(M^{-1}A\right) = \left\|M^{-1}A\right\|_\infty \left\|\left(M^{-1}A\right)^{-1}\right\|_\infty = \left\|A^{-1}M\right\|_\infty = \max_i \left\{ \sum_j \left| \left(a^{-1}\right)_{i,j} m_{j,j} \right| \right\}$$

$$\leq \max_k \{m_{k,k}\} \max_i \left\{ \sum_j \left|a^{-1}_{i,j}\right| \right\} = \|A\|_\infty \left\|A^{-1}\right\|_\infty = \kappa\left(A\right) \tag{88}$$

□

### 4.3.3 Proposed Implementation

Our method is based on two observations. Firstly, that the $(d+1)$-th derivative of a spline of degree $2d+1$ is a spline of degree $d$. We will note this spline as $T_d(x)$. In order to respect the boundary conditions described by equation (84) this spline must respect the following equations:

$$\partial_x^i T_d(k_0) = \partial_x^i T_d(k_{N_c}) = 0 \quad \forall 0 \leq i < d \tag{89}$$

Secondly, we observe that in order to respect equation (89), the first $d$ coefficients of the spline $T_d(x)$ must be equal to 0.

In order to use these observations, the coefficients of the spline $T_d(x)$ must be expressed as a function of the coefficients of the original spline $S_{2d+1}(x)$. The equations that we will use are derived from the following recurrence relationship [18]:

$$\partial_x b_{i,p}(x) = p\left(\frac{b_{i,p-1}(x)}{k_{i+p} - k_i} - \frac{b_{i+1,p-1}(x)}{k_{i+1+p} - k_{i+1}}\right) \tag{90}$$

Note that b-splines of lower order require fewer knots to describe the system. Thus if the b-splines are defined using equation (64) with the knots which describe a spline of degree $r = 2d+1$, the b-splines $b_{i,r-j}$ are outside the domain $[a,b]$ for all $i < j$ and $N - i < N - j$.

Let us use all this information to describe the derivative of a spline, $S_p(x)$, of degree $p$, defined on the knots which describe the spline $S_r(x)$ of degree $r$ such that $\partial_x^{(r-p)} S_r(x) = S_p(x)$. We obtain the following:

$$\partial_x S_p(x) = \sum_{i=r-p+1}^{N_c+r-1} c_i^{[r-p+1]} b_{i,p-1}(x) \tag{91}$$

$$= \sum_{i=r-p}^{N_c+r-1} c_i^{[r-p]} \partial_x b_{i,p}(x) \tag{92}$$

$$= \sum_{i=r-p}^{N_c+r-1} c_i^{[r-p]} p\left(\frac{b_{i,p-1}(x)}{k_{i+p} - k_i} - \frac{b_{i+1,p-1}(x)}{k_{i+1+p} - k_{i+1}}\right) \tag{93}$$

$$= c_{r-p}^{[r-p]} p \frac{b_{r-p,p-1}(x)}{k_r - k_{r-p}} - c_{N_c+r-1}^{[r-p]} p \frac{b_{N_c+r,p-1}(x)}{k_{N_c+r+p} - k_{N_c+r}} + \sum_{i=r-p+1}^{N_c+r-1} p \frac{c_i^{[r-p]} - c_{i-1}^{[r-p]}}{k_{i+p} - k_i} b_i^{p-1}(x) \tag{94}$$

where $c_i^{[j]}$ denotes the i-th coefficient of the spline describing the j-th derivative of the spline $S_{2d+1}(x)$ expressed on the full set of knots. This means that $c_i^{[j]} = 0 \quad \forall i < j$ as these coefficients describe b-splines which are either outside the domain or have no support. By comparing

| Quadrature degree | Spline degree | Number of points | Secrest | Intuitive | Recursive |
|---|---|---|---|---|---|
| 1 | 3 | 10 | $1.10 \cdot 10^{2}$ | $3.06 \cdot 10^{0}$ | $3.06 \cdot 10^{0}$ |
| | | 100 | $1.14 \cdot 10^{4}$ | $3.21 \cdot 10^{0}$ | $3.21 \cdot 10^{0}$ |
| | | 1000 | $1.14 \cdot 10^{6}$ | $3.21 \cdot 10^{0}$ | $3.21 \cdot 10^{0}$ |
| 2 | 5 | 10 | $9.75 \cdot 10^{3}$ | $2.99 \cdot 10^{1}$ | $6.57 \cdot 10^{0}$ |
| | | 100 | $1.07 \cdot 10^{8}$ | $2.97 \cdot 10^{1}$ | $7.85 \cdot 10^{0}$ |
| | | 1000 | $1.07 \cdot 10^{12}$ | $2.97 \cdot 10^{1}$ | $7.86 \cdot 10^{0}$ |
| 3 | 7 | 10 | $2.49 \cdot 10^{5}$ | $8.96 \cdot 10^{2}$ | $1.24 \cdot 10^{1}$ |
| | | 100 | $3.48 \cdot 10^{11}$ | $8.62 \cdot 10^{2}$ | $1.91 \cdot 10^{1}$ |
| | | 1000 | $6.15 \cdot 10^{18}$ | $8.62 \cdot 10^{2}$ | $1.92 \cdot 10^{1}$ |
| 4 | 9 | 10 | $2.94 \cdot 10^{6}$ | $4.38 \cdot 10^{4}$ | $1.96 \cdot 10^{1}$ |
| | | 100 | $6.05 \cdot 10^{14}$ | $4.12 \cdot 10^{4}$ | $4.68 \cdot 10^{1}$ |
| | | 1000 | $7.48 \cdot 10^{24}$ | $4.12 \cdot 10^{4}$ | $4.71 \cdot 10^{1}$ |
| 5 | 11 | 10 | $1.71 \cdot 10^{7}$ | $2.78 \cdot 10^{6}$ | $2.47 \cdot 10^{1}$ |
| | | 100 | $1.40 \cdot 10^{18}$ | $2.98 \cdot 10^{6}$ | $1.15 \cdot 10^{2}$ |
| | | 1000 | $5.17 \cdot 10^{24}$ | $2.98 \cdot 10^{6}$ | $1.16 \cdot 10^{2}$ |
| 6 | 13 | 10 | $5.03 \cdot 10^{7}$ | $2.30 \cdot 10^{8}$ | $4.62 \cdot 10^{1}$ |
| | | 100 | $3.92 \cdot 10^{22}$ | $3.08 \cdot 10^{8}$ | $2.82 \cdot 10^{2}$ |
| | | 1000 | $7.46 \cdot 10^{26}$ | $3.08 \cdot 10^{8}$ | $2.85 \cdot 10^{2}$ |
| 7 | 15 | 10 | $1.53 \cdot 10^{8}$ | $3.59 \cdot 10^{10}$ | $1.05 \cdot 10^{2}$ |
| | | 100 | $5.42 \cdot 10^{23}$ | $4.31 \cdot 10^{10}$ | $6.93 \cdot 10^{2}$ |
| | | 1000 | $2.02 \cdot 10^{26}$ | $4.31 \cdot 10^{10}$ | $7.03 \cdot 10^{2}$ |

Table 2: Table showing the conditioning of the matrix required to find the coefficients of a natural spline with equidistant knots on the domain $[-1, 1]$ calculated using three different methods.

coefficients we therefore obtain the following recursive relationship:

$$c_i^{[j]} = (r - j + 1)\frac{c_i^{[j-1]} - c_{i-1}^{[j-1]}}{k_{i+r-j+1} - k_i} \tag{95}$$

Equation (89) implies:

$$c_{d+1+i}^{[d+1]} = 0 \quad \forall 0 \leq i < d \tag{96}$$

$$c_{N_c+2d+1-i}^{[d+1]} = 0 \quad \forall 0 \leq i < d \tag{97}$$

A combination of equations (95), (96), and (97) therefore provides us with the equations required to fill the first and last $d$ rows of the interpolation matrix. The algorithm which fills these rows can be found in D.

#### 4.3.4 Precision Comparison

We therefore have 3 methods to compare. The first is the method proposed by Secrest [25], the second is the intuitive method described by equations (85)-(87), and the third is the recursive method described by equations (96), (86), and (97). As mentioned previously the condition number of the matrix required to calculate the quadrature coefficients is important. In general $k$ digits of accuracy of the solution of the matrix system are lost for a condition number $\kappa(A) = 10^k$[26] (starting from machine precision of $\sim 10^{-16}$). Therefore supposing we desire a final precision of $10^{-8}$ any matrix with a condition number $\kappa(A) > 10^8$ is unusable.

Table 2 shows the conditioning of the three different methods. We note that Secrest's method is very poorly conditioned, especially for large problems. For the requested precision we are limited to low quadrature degrees and few points. In contrast, the two b-spline based methods perform much better when the number of points is increased. These methods remove the dependency of the condition number on the problem size which allows much larger problems to be tackled effectively. Although the intuitive method performs significantly better than Secrest's method for a large number of points the condition number still increases rapidly with the degree. As a result

for the requested precision we would still be limited to 5th order quadrature. In contrast, the proposed recursive method does not suffer from this problem. While the condition number also increases with the degree it does so gradually. As a result, for the requested precision, quadrature coefficients can be calculated up to 20th order. The low condition number for orders which we are likely to use, means that we can be sure that the calculation of the quadrature coefficients is not introducing significant errors into the final quadrature calculation. The proposed recursive method will therefore be used for quadrature calculations in the rest of the paper.

# 5 Non-Uniform Convergence results

There are many different ways to define a set of non-uniform points and different choices for these points can lead to different convergence behaviour. The method used in this work is based on the following weight function [27]:

$$W(x) = \sqrt{1 + \alpha^2 \left(u_x(x)\right)^2} \tag{98}$$

where $u_x(x)$ is the gradient of the function $u(x)$ for which the points are being chosen, and $\alpha = 0.1$ is a constant. The coefficient $\alpha$ controls the ratio between the largest and smallest point density. The value of 0.1 was chosen as a good compromise in order to be sufficiently non-uniform to see the benefits, while keeping a reasonable point density in the rest of the domain.

In the case of the simulations, the function $u(x)$ is the distribution function $f(t, x, v)$. This function is defined in two dimensions and varies with time, therefore the role of $u_x(x)$ is modified slightly so that it is defined as follows:

$$u_x(x) = L_x \max_{t,v} |\partial_x f(t, x, v)| \tag{99}$$

where $L_x = (x_{N_x - 1} - x_0)$ is the length of the domain. It is used to normalise the derivative such that the severity of the slope is evaluated relative to the size of the domain. When determining the weight function in the velocity direction the roles of $x$ and $v$ are inverted.

The exact value of the distribution function is not known. It is therefore approximated from the results of a test simulation run with a small number of uniform points. The derivatives are approximated using a spline interpolation of the data. The resulting weight function can be quite noisy. This is due to multiple effects. Firstly, the simulation itself is noisy as the mesh is not sufficiently narrow to capture all the physics. This may in turn introduce noise into the spline approximation. Secondly the sampling rate may also introduce noise. As the position of the steep gradient changes rapidly at the beginning of the simulation, the position may have moved by more than the mesh step size between sampling points. If this is the case then the weight function may appear lower at an intermediate grid point for which the moment where the weight function was maximal was not sampled. In order to reduce these various sources of noise Fourier transforms are used to remove all frequencies corresponding to these error sources.

The method used in this paper is designed to generate cells of equal sizes locally. The weight function is therefore used to identify zones requiring a higher point density. The zones are selected using a cutoff of the normalised weight function $\hat{W}(x)$ defined as:

$$\hat{W}(x) = \frac{W(x) - 1}{\max_x \left[W(x) - 1\right]} \tag{100}$$

The domain is split at every point where the normalised weight function is equal to $0.2^n$, with $n \in \mathbb{Z}$.

Once the zones have been determined the cell size inside that zone is chosen to be inversely proportional to the largest value of $W(x)$ within the zone.

These cells are then used as the cells of the splines. The points on which the simulation evolves are the interpolation points of the splines, defined as the Greville abcissae (see equation (70)).

## 5.1 Splines

The convergence of the non-uniform splines is examined by interpolating a Maxwellian function defined as:

$$f(x) = \frac{1}{2\pi} \exp\left(-\frac{x^2}{2}\right) \tag{101}$$

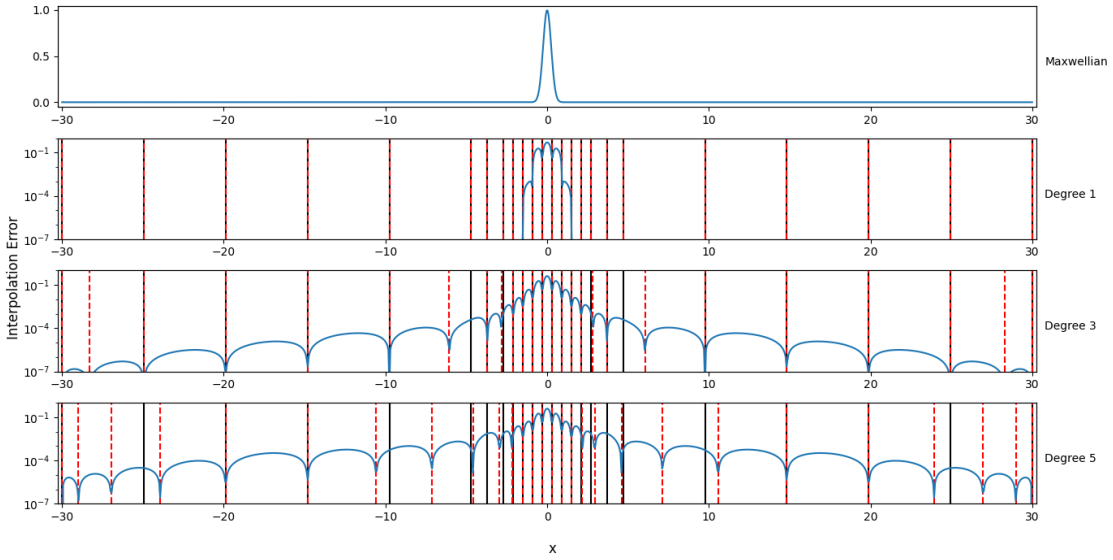



Figure 3: The error for the spline interpolation of equation (101), for splines of varying degrees. The knots of the spline are shown in black, while the interpolation points are shown by red dashed lines.

A Maxwellian function is an approximation of the distribution of particles in velocity space in a plasma, it is therefore an interesting test case when examining the convergence. In the simulations the domain $[-6, 6]$ will be used, however in this section the domain $[-30, 30]$ is considered. A larger domain is considered here as the gradients of a Maxwellian function in the domain $[-6, 6]$ are small compared to the size of the domain. This leads to quasi-equidistant points. This is not the case in the simulation, as larger gradients arise due to the interaction between the plasma and the wall as we will see in Section 6.

Figure 3 shows the error of the spline approximation of the function defined in equation (101) for various degrees of splines. The knots and Greville points are shown on the figure. We can see that the main factor of influence on the error is the proximity to an interpolation point. We also note that the L-$\infty$ error decreases as the degree of the spline increases, however using an overly high ordered spline is not necessarily advantageous as they can introduce small spurious oscillations in areas with little variation. In figure 3, this can be seen near the boundary where the error increases with the spline degree.

A spline of degree $d$ is said to be of order $d+1$, it is therefore important to confirm that the order agrees with our expectations. Figure 4 shows the convergence of the L2 norm of the interpolation error as a function of the number of cells. We see that the order increases as the degree increases, and that it is close to the expected value. However the non-equidistant points do have an effect on the convergence order. The order of convergence is slightly lower in the non-uniform case. Despite this, the improved choice of points leads to a smaller error unless a very large number of points are used.

Given that the error decreases rapidly for higher order splines it seems natural to use higher order splines in our simulations. However there is a trade-off between the error that can be obtained from the spline approximation and the time required to carry out the calculation. One way to reduce the temporal cost associated with higher order splines is to use effective parallelisation techniques. GPUs can allow us to carry out a large number of similar calculations in parallel. The use of GPUs for Vlasov equations solved using a backward semi-Lagrangian method on a spline basis with equidistant knots has already been studied by multiple groups [28, 29]. However as discussed in Section 4, non-uniform splines are less performant and may therefore potentially benefit more from the use of GPUs. This point will be discussed in Section 5.3.

## 5.2 Quadrature

The convergence of the quadrature scheme proposed in Section 4.3.1 is investigated using equations for which the exact integral is known. The first equation is the Maxwellian test function used in

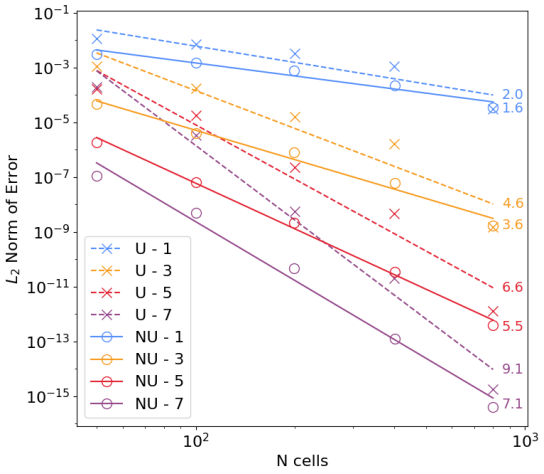
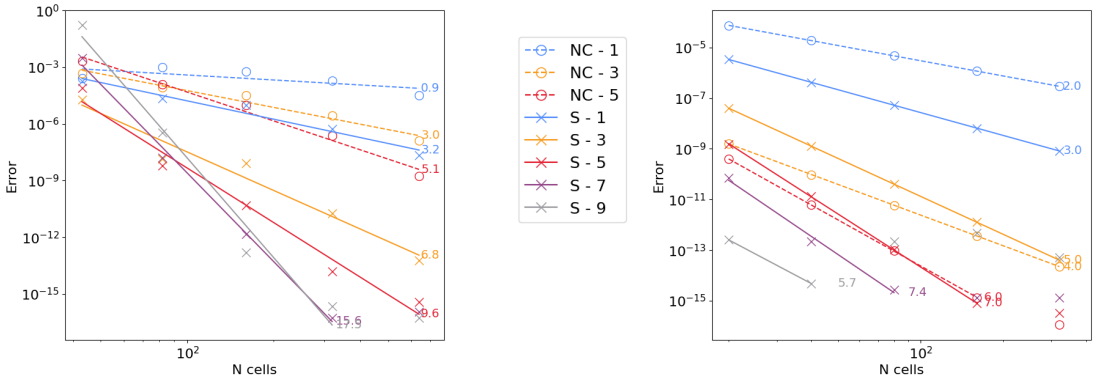



Figure 4: The convergence of the L2 norm of the interpolation error for a spline interpolation of a Maxwellian function for different degrees. "U - d" indicates uniform splines of degree $d$, while "NU - d" indicates non-uniform splines of degree $d$.




(a) Equation (102). (b) Equation (103).

Figure 5: The convergence of the error when solving Equations (103) and (102) with different degree Newton-Cotes (NC) or spline (S) quadrature schemes.

Section 5.1, the second is an integral suggested by Bailey et al. [30] for testing quadrature:

$$\int_{-30}^{30} \frac{1}{2\pi} \exp\left(-\frac{x^2}{2}\right) dx = \frac{1}{\sqrt{2\pi}} \operatorname{erf}\left(\frac{30}{\sqrt{2}}\right) \tag{102}$$

$$\int_0^1 \frac{\arctan\left(2+t^2\right)}{(1+t^2)\sqrt{2+t^2}} dx = 5\pi^2/96 \tag{103}$$

The quadrature is compared to three non-uniform Newton-Cotes schemes: the well-known trapezoid rule, Simpson's rule [31], and Boole-Villarceau's rule (see E for the non-uniform expression). The results can be seen in figure 5. We see that in the Maxwellian case examined in figure 5a where the natural boundary conditions provide a good representation of the actual boundary conditions, the order of convergence of the spline scheme is closer to the order of the underlying spline. As a result the spline scheme performs significantly better than the Newton-Cotes schemes.

In contrast, in figure 5b we see that the spline quadrature does not always perform better when integrating equation 103. However as it is much easier to implement higher-order non-uniform schemes with this method it is easy to use higher-order schemes to obtain improved errors.

## 5.3 Advection

There are two potential sources of errors for semi-Lagrangian advection: the calculation of the foot of the characteristic, and the evaluation of the function at that point. In this work we consider constant advection which allows an exact calculation of the foot of the characteristic. The error is therefore entirely described by the spline error. Figure 3 shows the error over the domain. We can see that the advection error will be smallest when the step is such that the feet of the characteristics are close to the knots.

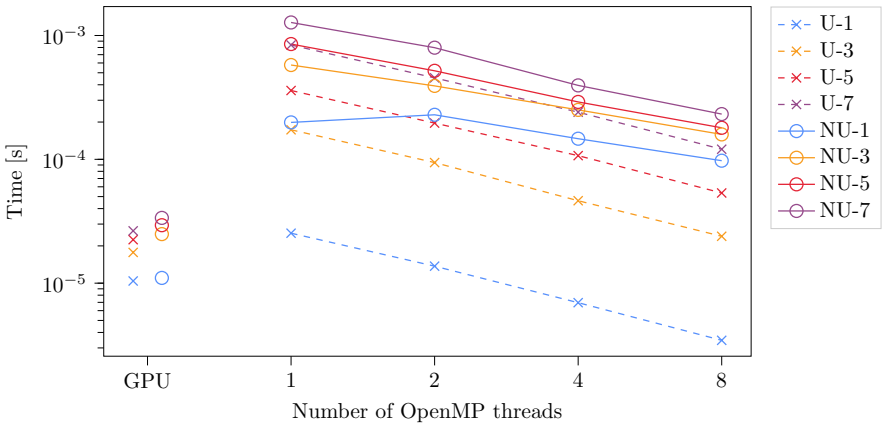



Figure 6: The time required to run an advection step for a grid with 2048 grid points for uniform (x) and non-uniform (∘) splines of various degrees on code accelerated with OpenMP for multi-threading or OpenACC for GPUs. Tests were run at the Centre de Calcul Intensif d'Aix Marseille.

As constant semi-Lagrangian advection consists only of spline interpolation and spline evaluation, the advection module can be used to examine the performance of the splines. Figure 6 shows the time taken for each advection step of the simulation for different parallelisation methods and spline degrees, for uniform and non-uniform splines. The OpenMP tests were run on a SkyLake processor with 192 GB of RAM. The OpenACC tests were run on a NVIDIA V100 GPU with 380GB of RAM. As expected, we note that non-uniform splines are significantly more costly than uniform splines, and that higher degree splines are also more costly. However we can also see that the scheme scales well when it is parallelised using OpenMP. This can allow the cost to be attenuated somewhat. We also see that GPUs present themselves as the natural solution to this problem. When using GPUs the increased cost is minimal and the total time required to run even potentially heavy simulations using non-uniform splines of degree 7 is still not significantly more costly than running uniform splines of degree 1 in serial.

## 5.4 Poisson

The implementation of the finite elements solver is tested using two manufactured solutions defined as linear combinations of Lorentzian functions. The right hand side of the equation is defined as:

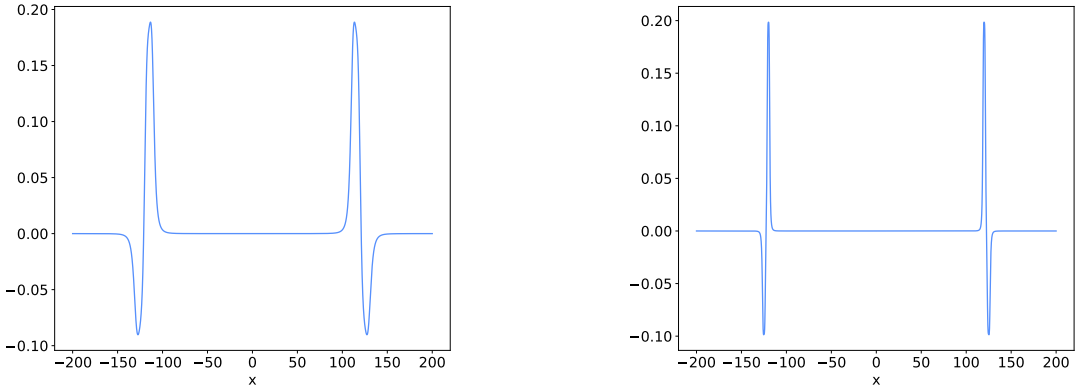
$$G(x, x0, w) = \frac{1}{w + (x - x0)^2} \tag{104}$$

$$\rho_1(x) = 200\,[G(x, -115, 1000) - G(x, -125, 2000) + G(x, 115, 1000) - G(x, 125, 2000)] \tag{105}$$

$$\rho_2(x) = G(x, -120, 5) - G(x, -125, 10) + G(x, 120, 5) - G(x, 125, 10) \tag{106}$$

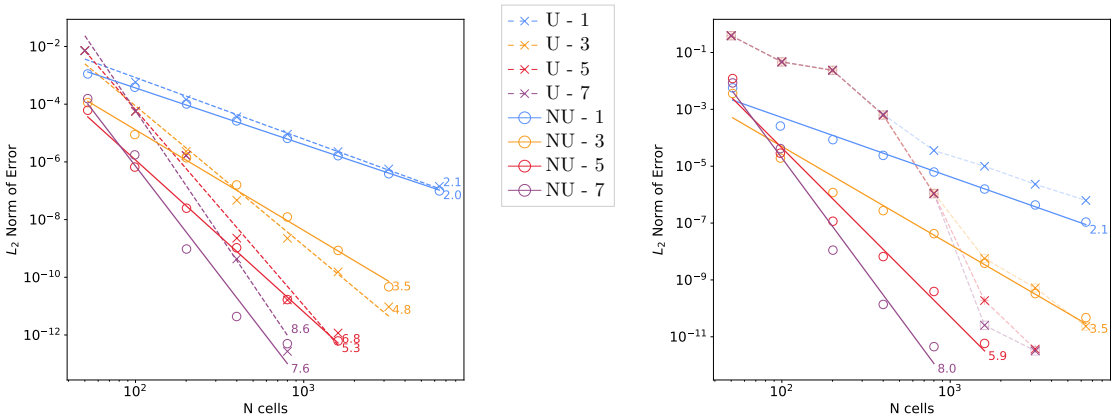
where $G(x, x0, w)$ is a Lorentzian function, and $x \in [-200, 200]$. This choice provides an equation with an analytical solution. The shape of $\rho_1(x)$ and $\rho_2(x)$ is chosen to be similar to the expected charge density profile (see Section 6). The shape can be seen in figure 7.

The results are shown in figure 8. In figure 8a, we see that once again there is some loss of convergence order compared to the theoretical value due to the use of non-equidistant knots. However the error still improves as the degree of the spline increases and machine precision is reached rapidly for higher order splines. In contrast, in figure 8b we see that uniform points struggle to handle very steep gradients. The order of convergence cannot be calculated for these




(a) Equation (105). (b) Equation (106).

Figure 7: The right-hand side of the test equation for the non-uniform Poisson solver.




(a) Equation (105). (b) Equation (106).

Figure 8: The convergence of the L2 norm of the error when solving equations (105) and (106) with the uniform (x) and non-uniform (∘) Poisson solver for different degrees of splines.

points. For the first few points we see a superposition of the calculated error. This occurs because there are not enough points at the peaks to accurately describe their shape. Thus, until there is a sufficient number of points, the convergence describes the sampling issues, not the Poisson scheme. On the other hand non-uniform points have no problem with such steepness as there is a higher point density in this area. These points therefore converge with the expected order.

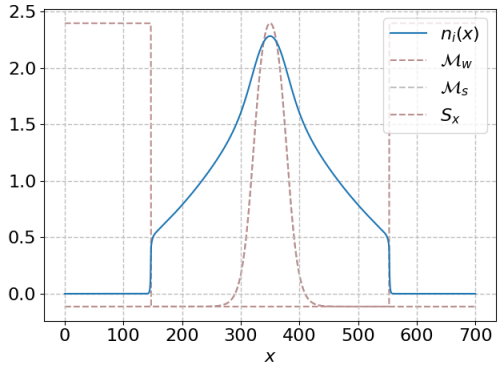
# 6 Plasma Simulations with Non-Equidistant Meshes

The tools described in sections 3 and 5 will now be used to run the simulation described in Section 2. The code used for this simulation is known as VOICE (Vlasov Open boundary Ion Coupling to Electrons). The domain used for the simulation is $[0, 700] \times [-6, 6]$. The parameters are described in table 3. The simulation was run for 20 000 time steps to a final time of $T = 2\,000$.

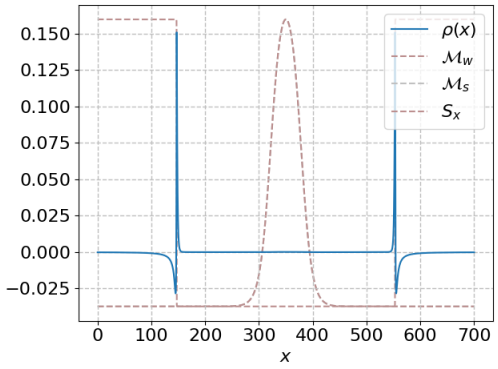
First we show some basic physical results showing that our simulation acts as expected. In

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $m_i$ | 400 | $m_e$ | 1 | $q_i$ | 1 | $q_e$ | -1 | $n_w$ | $10^{-11}$ | $T_w$ | 0.5 |
| $\nu_w$ | 0.1 | $\nu_d$ | 0.1 | $s_k$ | 0.1 | $T_k$ | 1 | $\nu_{ii}$ | 0.1 | $\nu_{ee}$ | 0.1 |
| $x_{Lw}$ | 147 | $x_{Rw}$ | 553 | $x_{Lk}$ | 322 | $x_{Rk}$ | 378 | $d_w$ | 0.1 | $d_k$ | 20 |
| $\Delta t$ | 0.1 | $x_0$ | 0 | $x_{N_x-1}$ | 700 | $v_0$ | -6 | $v_{N_v-1}$ | 6 | | |

Table 3: Definition of the constants used in the simulation described in Section 2.




(a) Ion density as defined in equation (4).

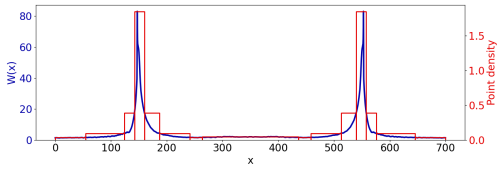



(b) Charge density as defined in equation (3).

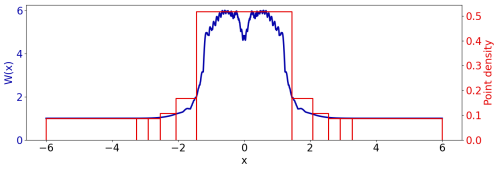
Figure 9: Physical results of a simulation with 2500 non-uniform points in the spatial dimension, 2000 non-uniform points in the velocity dimension at time t=2000, and splines of degree 3.

figure 9a we see that the plasma is correctly redistributed. The initial conditions describe a plasma with a homogeneous density throughout the domain; but the plasma density in the wall rapidly falls to zero as an equilibrium is established between particles injected by the source and particles absorbed by the wall. A higher density is seen in the central region where the particle source is placed.

In figure 9b we see the charge density profile. The quasi-neutrality is respected in the plasma region, but a positive charge develops in front of the wall forming the sheath region. This shows a build-up of ions approaching the wall which have been accelerated by the sheath. Electrons in the sheath are accelerated towards the plasma, however fast electrons are not sufficiently affected to change their trajectory, as a result we also see a build-up of electrons inside the wall which have been slowed as they exit the plasma region. These electrons have not yet been absorbed in order to preserve total charge. This is a direct consequence of the charge conservation imposed by equation (21) and explains the negative charge seen inside the wall. This equilibrium is discussed by Munschy et al [32] in a paper which focuses on the physical system.




(a) Spatial dimension.



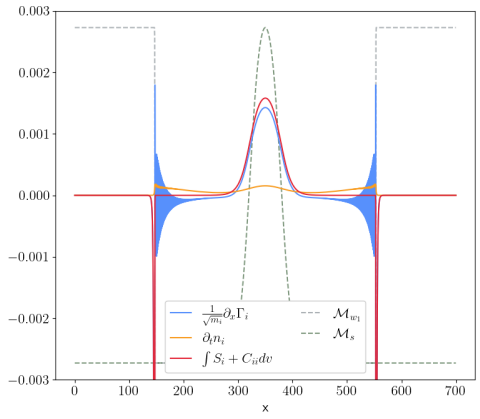

(b) Velocity dimension.

Figure 10: The weight function in each dimension calculated from a reference simulation with 1500 equidistant points in the spatial dimension, 500 equidistant points in the velocity dimension, and splines of degree 3, and the point density inferred from that function.

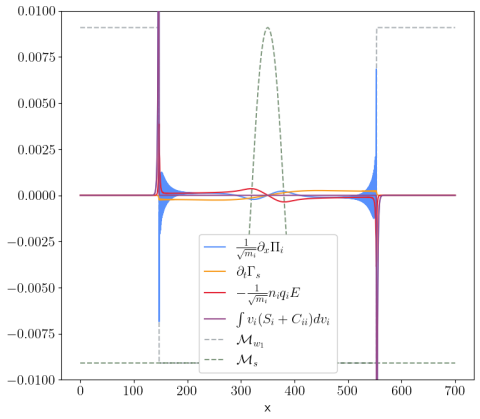
The non-uniform points chosen were calculated from a reference simulation run with equidistant points. This simulation was run with 1500 equidistant points in the spatial dimension and 500 equidistant points in the velocity dimension. The number of points used for this simulation was kept to a minimum, however the number of points in the spatial dimension is still relatively high. This is because the steep wall creates oscillations in under-resolved simulations which eventually lead to the appearance of NaN values in the simulation. The weight functions calculated for this reference simulation can be seen in figure 10.

As mentioned in Section 2.2 equations (32)-(34) are used to provide a method for evaluating the error which is decoupled from the equations used to solve the system. Here we only consider the error for the ions. Electrons move much faster so their resolution is time-limited. This makes it very costly to run simulations which would allow us to observe convergence in the spatial dimension. The different terms in the conservation equations can be seen in figure 11. We see that spatial derivatives close to the steep gradients describing the wall oscillate unphysically and the value at the plasma-wall transition is extremely large.

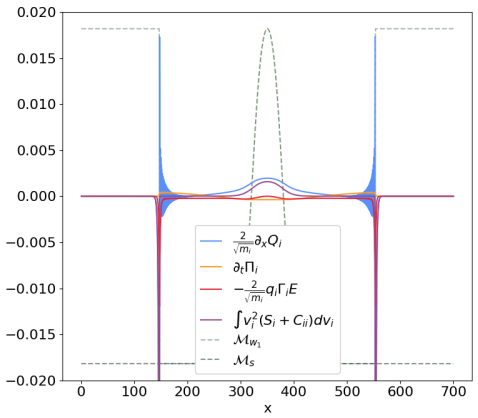
The error associated with these equations is shown in figure 12a. The error is concentrated around the wall transition region, with additional errors propagated into the plasma region due to oscillations. Non-equidistant points can be used to improve these errors.




(a) Density conservation, equation (32).




(b) Velocity conservation, equation (33).



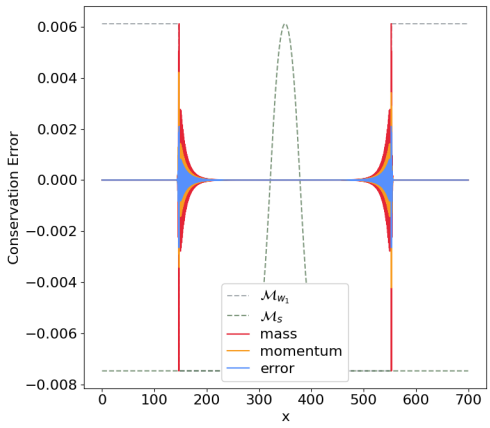

(c) Energy conservation, equation (34).

Figure 11: Terms from the conservation equations (32)- (34) for ions for a simulation with 1500 equidistant points in the spatial dimension, 500 equidistant points in the velocity dimension at time t=2000, and splines of degree 3. The axis of ordinates is truncated to illustrate the shape of the functions.

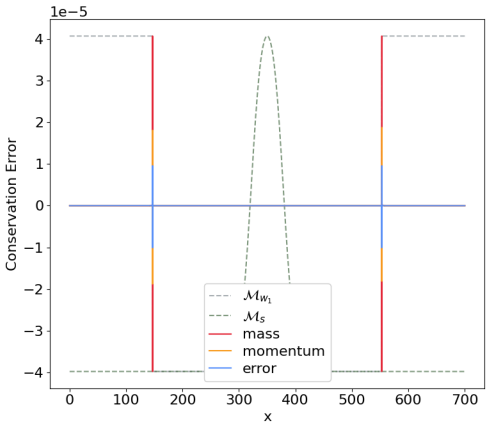
Figure 13 shows the variation of the conservation errors as the number of uniform or non-uniform points in the spatial dimension is varied. We see that the error converges much more rapidly when non-uniform points are used, rapidly reaching an error equivalent to the truncation error discussed in Section A. The conservation errors for a case refined with non-equidistant points can be seen in figure 12b. We see that the large unphysical oscillations have been removed from the simulation.

The conservation represents fluid quantities, however the code described here is a kinetic code. Such codes contain more details which can be hard to model. We will therefore also examine a second non-fluid criteria: the presence of negative values in the plasma. A distribution function can never be negative, however our numerical methods do not impose the positivity. Spurious oscillations caused by steep gradients can easily lead to the presence of negative values. Figure 14 shows the convergence of the negative values in the plasma. The plasma region is identified as the central region between 2 points identifying the start of the wall. The start of the wall is defined as the place where the charge density is equal to zero. This is calculated for the most refined non-uniform case and the same positions are used to identify the region for all other cases.

The simulation can only be used for physical studies when there are no more negative values in the plasma region. We see that for non-uniform points this occurs at Nx=2500. However the convergence for uniform points is much slower. It was not feasible to run a large enough




(a) Simulation with 1500 equidistant points in the spatial dimension and 500 equidistant points in the velocity dimension.




(b) Simulation with 2500 non-equidistant points in the spatial dimension and 2000 equidistant points in the velocity dimension.

Figure 12: The error associated with the conservation of density (equation (32)), velocity (equation (33)), and energy (equation (34)) for ions at time t=2000 with splines of degree 3.

| | L2-Norm of Convergence Error for Ions | | |
|---|---|---|---|
| Degree | Density | Velocity | Energy |
| 3 | $1.46 \cdot 10^{-8}$ | $3.66 \cdot 10^{-8}$ | $1.16 \cdot 10^{-7}$ |
| 5 | $2.06 \cdot 10^{-9}$ | $5.69 \cdot 10^{-9}$ | $2.00 \cdot 10^{-8}$ |
| 7 | $2.53 \cdot 10^{-9}$ | $6.73 \cdot 10^{-9}$ | $2.28 \cdot 10^{-8}$ |

Table 4: Comparison of conservation errors for different degrees of splines for a simulation with 1000 cells in the spatial dimension and 503 cells in the velocity dimension.

simulation to remove all negative points while using equidistant points, however if it is assumed that the necessary spatial resolution would be equal to the smallest resolution in the successful non-equidistant case, we calculate that 23 217 points would be necessary. This is more than 9 times as many points as in the non-uniform case.

In Section 5 we showed that higher order splines can lead to lower errors and that the cost of using these splines is minimal when using GPU acceleration. It is therefore logical to try to use higher order splines in our simulation. Table 4 shows the results for a case with 1000 cells in the spatial dimension and 503 cells in the velocity dimension. Tests cannot be run with degree 1 splines as the collisions use the second derivative of the spline so the problem would not be fully defined. We see that the $L_2$ norm of the conservation errors decreases as the spline order increases from 3 to 5 however the difference stagnates as we increase to degree 7. This is due to the oscillations that are introduced by high order splines (as seen in figure 3).

For negative values in the plasma, the higher degree introduces oscillations which quickly cause problems, as can be seen in table 5. Negative values appear in the plasma region of both the ion and electron distribution function for degrees larger than 3. For electrons the convergence usually occurs more quickly as they move towards a stable configuration more quickly. This can be seen in figure 14b. However this is not the case for higher order splines. As a result more points are required to respect the positivity of the distribution function when higher order splines are used.

# 7 Conclusion

In this paper we have shown that a judicious choice of non-uniform points can be used to reduce the memory constraints for sheath simulations with steep gradients by 89%. Without this reduction it would be impossible to run the equivalent simulation with uniform points on a GPU node of the Centre de Calcul Intensif d'Aix Marseille (380GB of RAM). These improvements have therefore permitted the physical study conducted by Y. Munschy et al [32]. Furthermore although these

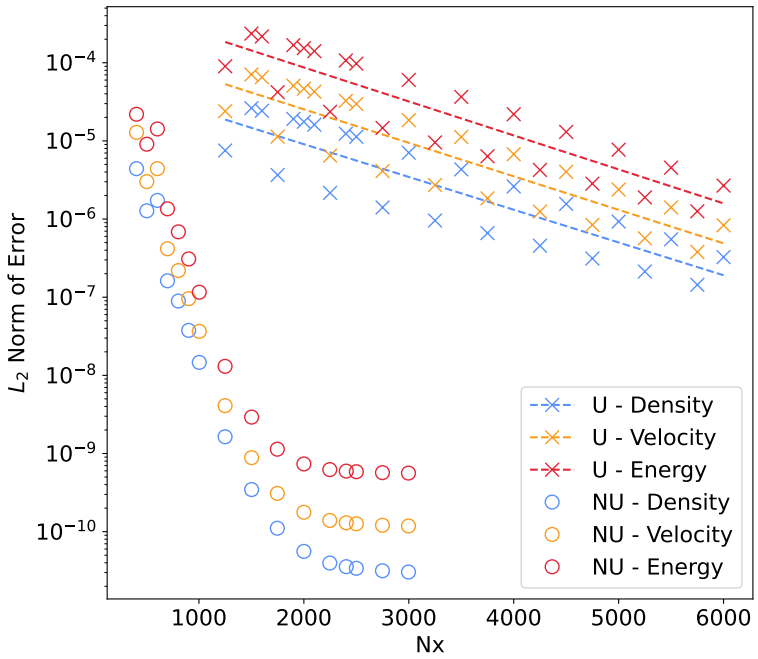



Figure 13: Conservation errors as described in Section 2.2 at time t=2000 with 2000 points in the velocity dimension. "U - X" indicates the X conservation for uniform splines of degree 3, while "NU - X" indicates the X conservation for non-uniform splines of degree 3. The saturation of the error at $10^{-9}$ is expected due to the truncation error discussed in Section A.

| Degree | Smallest negative value in the distribution function restricted to the plasma region | |
|---|---|---|
| | Ions | Electrons |
| 3 | 0 | 0 |
| 5 | 0 | $-5.59 \cdot 10^{-10}$ |
| 7 | 0 | $-6.84 \cdot 10^{-9}$ |

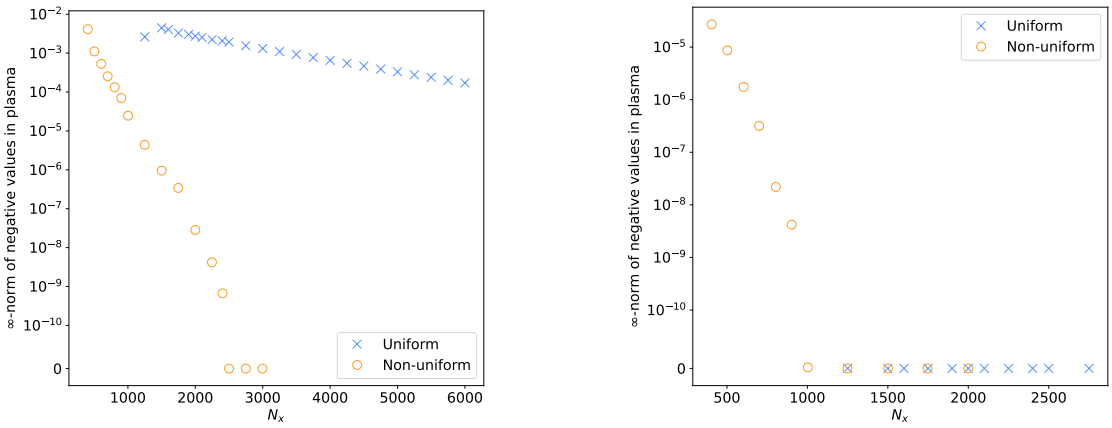
Table 5: Comparison of negative values in the distribution function for different degrees and different species.

are already large simulations, they do not yet represent the real dimensions of the domain we are simulating. These improvements will allow the simulation domain to be extended to simulate a situation closer to reality.

Different degrees of spline were compared for the main numerical schemes used in the simulation. It was found that high-order schemes converge faster but can introduce spurious oscillations. This means that high-order schemes are not adapted to kinetic simulations as more points are required to ensure that the distribution function remains positive. Cubic splines seem to be a good compromise, as they converge quickly without adding too many oscillations, which allows the distribution function to preserve its positivity with a limited number of points.

The fluid convergence results indicate that high-order schemes may be more useful for simulating regions whose behaviour is sufficiently described by a fluid model. In these areas the kinetic properties, such as the positivity of the distribution function, are less important, so they can be permitted to oscillate as long as this does not have a negative effect on the fluid quantities.

The numerical cost of higher-order non-uniform spline schemes was also studied. Higher-order spline schemes are slower than lower-order spline schemes, and non-uniform schemes are slower than uniform schemes. For cubic splines on GPUs, non-uniform schemes are 30% slower than uniform schemes (see figure 6). The reduction in the number of points when using non-uniform schemes in place of uniform schemes must therefore be at least 30% for the simulation to run in a similar time




(a) Ions. (b) Electrons.

Figure 14: The magnitude of the most negative values in the distribution function restricted to the plasma region for simulations with 2000 points in the velocity dimension and splines of degree 3.

frame on the same machine. In contrast, on 8 OpenMP threads, non-uniform schemes are 85% slower than uniform schemes (see figure 6). The reduction in points must therefore be at least 85% for the simulation to run in a similar time frame. In the simulations described, an 89% reduction in the number of points was obtained for equivalent results. This means that there would be very limited speed gains when moving to non-uniform points, if these simulations are run on OpenMP. However the parallelisation methods used by GPUs seem more effective, therefore allowing non-uniform simulations to run 5.5 times faster than uniform simulations providing equivalent results. GPU parallelisation is therefore an important tool when accelerating non-uniform schemes based on splines.

These simulations provide a clear path to the implementation of non-uniform points in the GYSELA code. The memory gains there should be even larger than those seen in this paper as small reductions in one dimension lead to large reductions in the total number of points in a 5D simulation.

## Acknowledgements

The authors would like to thank Yaman Güçlü for the many fruitful discussions concerning splines. Centre de Calcul Intensif d'Aix-Marseille is acknowledged for granting access to its high performance computing resources. This work was carried out using splines from the SeLaLib project [33]. The project has received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 824158 (EoCoE-II).

## References

[1] G. Dif-Pradalier, P. Ghendrih, Y. Sarazin, E. Caschera, F. Clairet, Y. Camenen, P. Donnel, X. Garbet, V. Grandgirard, Y. Munschy, L. Vermare, and F. Widmer, "Transport barrier onset and edge turbulence shortfall in fusion plasmas," *Communications Physics*, 2022. in press.

[2] D. Coulette and G. Manfredi, "An eulerian vlasov code for plasma-wall interactions," *Journal of Physics: Conference Series*, vol. 561, p. 012005, nov 2014.

[3] V. Grandgirard, J. Abiteboul, J. Bigot, T. Cartier-Michaud, N. Crouseilles, G. Dif-Pradalier, C. Ehrlacher, D. Esteve, X. Garbet, P. Ghendrih, G. Latu, M. Mehrenberger, C. Norscini, C. Passeron, F. Rozar, Y. Sarazin, E. Sonnendrücker, A. Strugarek, and D. Zarzoso, "A 5d gyrokinetic full-f global semi-lagrangian code for flux-driven ion turbulence simulations," *Computer Physics Communications*, vol. 207, pp. 35–68, 2016.

[4] G. Manfredi and F. Valsaque, "Vlasov simulations of plasma-wall interactions in a weakly collisional plasma," *Computer Physics Communications*, vol. 164, no. 1, pp. 262–268, 2004. Proceedings of the 18th International Conferene on the Numerical Simulation of Plasmas.

[5] G. Manfredi, S. Hirstoaga, and S. Devaux, "Vlasov modelling of parallel transport in a tokamak scrape-off layer," *Plasma Physics and Controlled Fusion*, vol. 53, p. 015012, nov 2010.

[6] Badsi, M., Mehrenberger, M., and Navoret, L., "Numerical stability of a plasma sheath," *ESAIM: ProcS*, vol. 64, pp. 17–36, 2018.

[7] B. Afeyan, F. Casas, N. Crouseilles, A. Dodhy, E. Faou, M. Mehrenberger, and E. Sonnendrücker, "Simulations of kinetic electrostatic electron nonlinear (keen) waves with variable velocity resolution grids and high-order time-splitting," *The European Physical Journal D*, vol. 68, p. 295, Oct 2014.

[8] G. Dif-Pradalier, P. Diamond, V. Grandgirard, Y. Sarazin, J. Abiteboul, X. Garbet, P. Ghendrih, G. Latu, A. Strugarek, S. Ku, , and C. Chang, "Neoclassical physics in full distribution function gyrokinetics," *Phys. Plasmas*, vol. 18, p. 062309, 2011.

[9] P. L. Bhatnagar, E. P. Gross, and M. Krook, "A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems," *Phys. Rev.*, vol. 94, pp. 511–525, May 1954.

[10] Y. Sarazin, V. Grandgirard, J. Abiteboul, S. Allfrey, X. Garbet, P. Ghendrih, G. Latu, A. Strugarek, G. Dif-Pradalier, P. Diamond, S. Ku, C. Chang, B. McMillan, T. Tran, L. Villard, S. Jolliet, A. Bottino, and P. Angelino, "Predictions on heat transport and plasma rotation from global gyrokinetic simulations," *Nuclear Fusion*, vol. 51, p. 103023, sep 2011.

[11] A. Paredes, H. Bufferand, G. Ciraolo, F. Schwander, E. Serre, P. Ghendrih, and P. Tamain, "A penalization technique to model plasma facing components in a tokamak with temperature variations," *Journal of Computational Physics*, vol. 274, pp. 283–298, 2014.

[12] E. Caschera, G. Dif-Pradalier, P. Ghendrih, V. Grandgirard, Y. Asahi, N. Bouzat, P. Donnel, X. Garbet, G. Latu, C. Passeron, and Y. Sarazin, "Immersed boundary conditions in global, flux-driven, gyrokinetic simulations," *Journal of Physics: Conference Series*, vol. 1125, p. 012006, nov 2018.

[13] G. Strang, "On the construction and comparison of difference schemes," *SIAM Journal on Numerical Analysis*, vol. 5, pp. 506–517, 9 1968.

[14] E. Sonnendrücker, J. Roche, P. Bertrand, and A. Ghizzo, "The semi-lagrangian method for the numerical resolution of the vlasov equation," *Journal of Computational Physics*, vol. 149, no. 2, pp. 201–220, 1999.

[15] A. Vermeulen, R. Bartels, and G. Heppler, "Integrating products of b-splines," *Siam Journal on Scientific and Statistical Computing*, vol. 13, 07 1992.

[16] G. Farin, "Chapter 10 - b-splines," in *Curves and Surfaces for Computer-Aided Geometric Design (Third Edition)*, pp. 157–187, Boston: Academic Press, third edition ed., 1993.

[17] N. M. Patrikalakis and T. Maekawa, *Representation of Curves and Surfaces*, pp. 1–33. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002.

[18] G. Hämmerlin and K.-H. Hoffman, *Splines*, pp. 229–271. New York, NY: Springer New York, 1991.

[19] W. J. Gordon and R. F. Riesenfeld, "B-spline curves and surfaces," in *Computer Aided Geometric Design* (R. E. BARNHILL and R. F. RIESENFELD, eds.), pp. 95–126, Academic Press, 1974.

[20] C. De Boor, "On bounding spline interpolation," *Journal of Approximation Theory*, vol. 14, no. 3, pp. 191–203, 1975.

[21] C. De Boor and C. De Boor, *A practical guide to splines*, vol. 27. springer-verlag New York, 1978.

[22] D. E. Knuth, *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[23] C. d. Boor, T. Lyche, and L. L. Schumaker, "On calculating with b-splines ii. integration," in *Numerische Methoden der Approximationstheorie/Numerical Methods of Approximation Theory*, pp. 123–146, Springer, 1976.

[24] I. J. Schoenberg, "Spline interpolation and best quadrature formulae," *Bulletin of the American Mathematical Society*, vol. 70, no. 1, pp. 143 – 148, 1964.

[25] D. Secrest, "Best approximate integration formulas and best error bounds," *j-MATH-COMPUT*, vol. 19, pp. 79–83, 04 1965.

[26] E. W. Cheney and D. R. Kincaid, *Numerical mathematics and computing*. Cengage Learning, 2012.

[27] M. Farrashkhalvat and J. Miles, "6 - variational methods and adaptive grid generation," in *Basic Structured Grid Generation* (M. Farrashkhalvat and J. Miles, eds.), pp. 152–179, Oxford: Butterworth-Heinemann, 2003.

[28] Mehrenberger, M., Steiner, C., Marradi, L., Crouseilles, N., Sonnendrücker, E., and Afeyan, B., "Vlasov on gpu," *ESAIM: Proc.*, vol. 43, pp. 37–58, 2013.

[29] G. Latu, "Fine-grained parallelization of a vlasov-poisson application on gpu," in *Euro-Par 2010 Parallel Processing Workshops* (M. R. Guarracino, F. Vivien, J. L. Träff, M. Cannatoro, M. Danelutto, A. Hast, F. Perla, A. Knüpfer, B. Di Martino, and M. Alexander, eds.), (Berlin, Heidelberg), pp. 127–135, Springer Berlin Heidelberg, 2011.

[30] D. Bailey, K. Jeyabalan, and S. Li, "A comparison of three high-precision quadrature schemes," *Experimental Mathematics*, vol. 14, 01 2011.

[31] N. Shklov, "Simpson's rule for unequally spaced ordinates," *The American Mathematical Monthly*, vol. 67, no. 10, pp. 1022–1023, 1960.

[32] Y. Munschy, E. Bourne, G. Dif-Pradalier, Y. Sarazin, V. Grandgirard, and P. Ghendrih, "Kinetic plasma-wall interaction using immersed boundary conditions." to be submitted, 2022.

[33] "Semi-lagrangian library." https://selalib.github.io/selalib/.

[34] L. Piegl and W. Tiller, *The NURBS book*. Springer Science & Business Media, 1996.

# A Expected Conservation Error

When calculating the error for the conservation equations (32) - (34) some of the error will be due to the truncation of the domain and is therefore unavoidable. The distribution function $f_s(t, x, v_s)$ can be approximated by a Maxwellian with density $n_s(t, x) = 1$, and $T_s(t, x) = 1$:

$$f_s(t, x, v_s) = \exp\left(-\frac{v_s^2}{2}\right) \tag{107}$$

If the domain is truncated in the velocity dimension to $[-v_T, v_T]$, the error due to this truncation is:

$$\varepsilon_i = C_i \int_{-\infty}^{\infty} v_s^i \exp\left(-\frac{v_s^2}{2}\right) dv_s - C_i \int_{-v_T}^{v_T} v_s^i \exp\left(-\frac{v_s^2}{2}\right) dv_s \tag{108}$$

$$C_i = \int_{-\infty}^{\infty} v_s^i \exp\left(-\frac{v_s^2}{2}\right) dv_s \tag{109}$$

where $\varepsilon_i$ is the truncation error for the $i$-th moment of the distribution function, and $C_i$ is a normalisation coefficient.

| Expected error | Required $v_T$ |
|---|---|
| $10^{-15}$ | 8.13 |
| $10^{-14}$ | 7.86 |
| $10^{-13}$ | 7.56 |
| $10^{-12}$ | 7.26 |
| $10^{-11}$ | 6.94 |
| $10^{-10}$ | 6.60 |
| $10^{-9}$ | 6.25 |
| $10^{-8}$ | 5.88 |
| $10^{-7}$ | 5.49 |
| $10^{-6}$ | 5.07 |

Table 6: $v_T$ necessary to attain an expected error assuming that the distribution function is Maxwellian, as defined in equation (110).

Thanks to symmetry properties there is no truncation error for an even $i$. Thanks to the normalisation, the error for an odd $i$ is always the same. The truncation error is expressed as follows:

$$\varepsilon = \sqrt{2\pi}\,\mathrm{erf}\left(\frac{\infty}{\sqrt{2}}\right) - \sqrt{2\pi}\,\mathrm{erf}\left(\frac{a}{\sqrt{2}}\right) = \sqrt{2\pi}\left(1 - \mathrm{erf}\left(\frac{a}{\sqrt{2}}\right)\right) \tag{110}$$

Table 6 shows the truncation error for different cut-off values $v_T$.

# B B-Spline shape

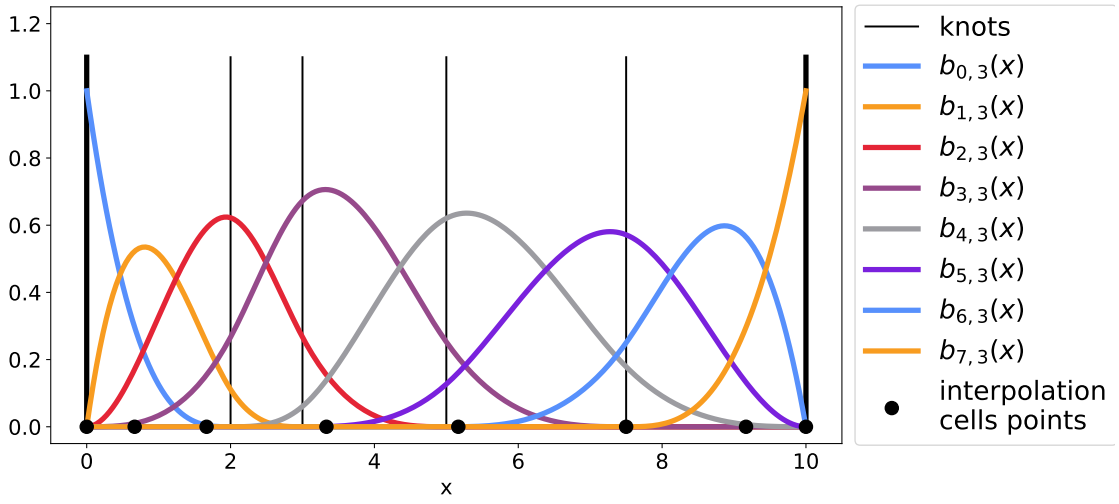



Figure 15: The shape of the bsplines and the position of the interpolation points for a set of non-equidistant knots with the Greville boundary conditions described in Section 4

# C B-Spline Evaluation Algorithms

In this section the algorithms required for evaluating basis splines given different constraints are shown. The most restrictive but optimised version is the case of 4th order uniform b-splines as shown in Section C.3. The algorithms used in the simulations are arbitrary order and the non-uniform and uniform versions are presented in Sections C.1 and C.2 respectively.

## C.1 Non-Uniform B-Splines of Arbitrary Order

This algorithm is the one proposed in Algorithm A2.2 in The NURBS book[34].

**Algorithm 1** Algorithm for calculating the value of arbitrary order non-uniform b-splines

**Input:** $x$ : Evaluation point
**Input:** $d$ : Spline degree
**Input:** $k$ : The knots of the spline
**Input:** $i^\star$ : The index of the last knot before the evaluation point
  $v_1^0 = 1$
  **for** $j = 1, \dots, d$ **do**
    $l_k = x - k_{i^\star - j - 1}$
    $r_k = k_{i^\star - j} - x$
    $s_0 = 0$
    **for** $t = 1, \dots, j$ **do**
      $v_t^j = s_{t-1} + r_t \frac{v_t^{j-1}}{r_t - l_{j-t}}$
      $s_t = l_{j-t} \frac{v_t^{j-1}}{r_t - l_{j-t}}$
    **end for**
    $v_t^j = s_k$
  **end for**
  **for** $j = 1, \dots, d$ **do**
    $B_{i^\star + j - 1, d}(x) = v_j^d$
  **end for**

The number of FLOPs required for this algorithm is counted as follows:

$$FLOPs = \sum_{j=1}^{d} 2 + \sum_{t=1}^{j} 4 + DIV = 2d + 8 \sum_{j=1}^{d} j = 4d^2 + 6d \tag{111}$$

where the term $\frac{v_t^{j-1}}{r_t - l_{j-t}}$ is precalculated to avoid repetition

## C.2 Uniform B-Splines of Arbitrary Order

This algorithm is a slightly optimised version of the algorithm in Section C.1 using the fact that the knots are equidistant.

**Algorithm 2** Algorithm for calculating the value of arbitrary order uniform b-splines

**Input:** $x$ : Evaluation point
**Input:** $d$ : Spline degree
**Input:** $i^\star$ : The index of the last knot before the evaluation point
**Input:** $k_{i^\star}$ : The position of the last knot before the evaluation point
**Input:** $\Delta x$ : The step between consecutive knots
  $v_1^0 = 1$
  $o = \frac{k_{i^\star} - x}{\Delta x}$
  **for** $j = 1, \dots, d$ **do**
    $s_0 = 0$
    **for** $r = 1, \dots, j - 1$ **do**
      $v_r^j = s_{r-1} + (o + r) \frac{v_r^{j-1}}{j}$
      $s_r = (j - (o + r)) \frac{v_r^{j-1}}{j}$
    **end for**
    $v_r^j = s_k$
  **end for**
  **for** $j = 1, \dots, d$ **do**
    $B_{i^\star + j - 1, d}(x) = v_k^d$
  **end for**

The number of FLOPs required for this algorithm is counted as follows:

$$FLOPs = 1 + DIV + \sum_{j=1}^{d} \sum_{t=1}^{j} 5 + DIV = 5 + \sum_{j=1}^{d} 9j = 5 + 9\frac{d(d+1)}{2} \tag{112}$$

where the terms $\frac{v_r^{j-1}}{j}$ and $o+r$ are calculated to avoid repetition.

### C.3 4th order Uniform B-Splines

$$o = (x - x_{i^\star})\, i\_dx \tag{113}$$
$$b = 1 - o \tag{114}$$
$$B_{i^\star-3,3}(x) = b^3 \tag{115}$$
$$B_{i^\star-2,3}(x) = 1 + 3(1 - b^2(2 - b)) \tag{116}$$
$$B_{i^\star-1,3}(x) = 1 + 3(1 - o^2(2 - o)) \tag{117}$$
$$B_{i^\star,3}(x) = o^3 \tag{118}$$

where $i\_dx = \frac{1}{dx}$ is the inverse of the step between consecutive knots which can be calculated in advance.

## D Spline Natural Boundary Conditions Algorithm

In this section we detail the algorithm used to express the elements of the matrix which describe equations (96) and (97) for all $0 \leq i < d$:

$$c_{d+1+i}^{[d+1]} \left( c_{i-d-1}^{[0]}, \ldots, c_i^{[0]} \right) = 0 \tag{119}$$
$$c_{N_c+2d+1-i}^{[d+1]} \left( c_{N_c+d-i}^{[0]}, \ldots, c_{N_c+2d+1-i}^{[0]} \right) = 0 \tag{120}$$

given the definition of $c_i^{[j]}$ from equation (95):

$$c_i^{[j]} = (r - j + 1) \frac{c_i^{[j-1]} - c_{i-1}^{[j-1]}}{k_{i+r-j+1} - k_i} \tag{121}$$

The algorithm which constructs the necessary matrix is based on the fact that equation (95) can be expressed as a matrix equation:

$$c_i^{[j]} = \begin{bmatrix} -\frac{(r-j+1)}{k_{i+r-j+1}-k_i} & \frac{(r-j+1)}{k_{i+r-j+1}-k_i} \end{bmatrix} \begin{pmatrix} c_{i-1}^{[j-1]} \\ c_i^{[j-1]} \end{pmatrix} \tag{122}$$

$$= \frac{(r-j+1)}{k_{i+r-j+1} - k_i} \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} -\frac{(r-j+2)}{k_{i+r-j+1}-k_{i-1}} & \frac{(r-j+2)}{k_{i+r-j+1}-k_{i-1}} \\ -\frac{(r-j+2)}{k_{i+r-j+2}-k_i} & \frac{(r-j+2)}{k_{i+r-j+2}-k_i} \end{bmatrix} \begin{pmatrix} c_{i-2}^{[j-2]} \\ c_{i-1}^{[j-2]} \\ c_i^{[j-2]} \end{pmatrix} \tag{123}$$

The algorithm therefore loops over $j$ calculating the line of the matrix representing:

$$c_i^{[d+1]} \left( c_{i-j}^{[d+1-j]}, \ldots, c_i^{[d+1-j]} \right)$$

until $j = d + 1$.

The algorithm is as follows:

**Algorithm 3** Algorithm for calculating the line matrix

**Input:** $i$ : Index of the coefficient to be expressed
**Input:** $d$ : Degree of the spline which should be interpolated exactly

**Output:** $M^d$ : $1 \times (d+1)$ matrix expressing $c_i^{[d+1]}$ as a linear combination of $c_{i-d-1}^{[0]}, \ldots, c_i^{[0]}$

$s = \frac{d+1}{k_{i+d+1} - k_i}$
$M_0^0 = -s$
$M_1^0 = s$
**for** $j = 1, \ldots, d$ **do**
  $s = \frac{d+j+1}{k_{i+d+1} - k_{i-j}}$
  $v = M_0^{j-1} s$
  $M_0^j = -v$
  **for** $c = 1, \ldots, j$ **do**
    $s = \frac{d+j+1}{k_{i+c+d+1} - k_{i-j+c}}$
    $M_c^j = v - M_c^{j-1} s$
    $v = M_c^{j-1} s$
  **end for**
  $M_{j+1}^j = v$
**end for**
$L_1 = 0$
**for** $j = 0, \ldots, d+1$ **do**
  $L_1 = L_1 + \left| M_j^d \right|$
**end for**
$M^d = \frac{M^d}{L_1}$

# E Boole-Villarceau's Rule

The non-equidistant form of Boole-Villarceau's rule has the following form:

$$\int f(x)dx \approx \sum_{i=0}^{N/4-1} (\alpha_i f_{4i} + \beta_i f_{4i+1} + \gamma_i f_{4i+2} + \zeta_i f_{4i+3} + \eta_i f_{4i+4}) \tag{124}$$

The coefficients are defined as follows:

$$\begin{aligned}
\alpha =& \frac{\sum_{i=1}^4 h_i}{60h_1\left(h_1+h_2\right)\left(h_1+h_2+h_3\right)} \left(12h_1^3 + 21h_1^2h_2 + 6h_1^2h_3 - 9h_1^2h_4 + 6h_1h_2^2 \right. \\
& + 2h_1h_2h_3 - 8h_1h_2h_4 - 4h_1h_3^2 + 2h_1h_3h_4 + 6h_1h_4^2 - 3h_2^3 - 4h_2^2h_3 \\
& \left. + h_2^2h_4 + h_2h_3^2 + 2h_2h_3h_4 + h_2h_4^2 + 2h_3^3 + h_3^2h_4 - 4h_3h_4^2 - 3h_4^3\right) \qquad (125) \\
\beta =& \frac{\left(\sum_{i=1}^4 h_i\right)^3}{60h_1h_2\left(h_2+h_3\right)\left(h_2+h_3+h_4\right)} \left(3h_1^2 + 6h_1h_2 + h_1h_3 - 4h_1h_4 \right. \\
& \left. + 3h_2^2 + h_2h_3 - 4h_2h_4 - 2h_3^2 + h_3h_4 + 3h_4^2\right) \qquad (126) \\
\gamma =& -\frac{\left(\sum_{i=1}^4 h_i\right)^3}{60h_2h_3\left(h_1+h_2\right)\left(h_3+h_4\right)} \left(3h_1^2 + h_1h_2 + h_1h_3 - 4h_1h_4 - 2h_2^2 \right. \\
& \left. - 4h_2h_3 + h_2h_4 - 2h_3^2 + h_3h_4 + 3h_4^2\right) \qquad (127) \\
\zeta =& \frac{\left(\sum_{i=1}^4 h_i\right)^3}{60h_3h_4\left(h_2+h_3\right)\left(h_1+h_2+h_3\right)} \left(3h_1^2 + h_1h_2 - 4h_1h_3 - 4h_1h_4 - 2h_2^2 \right. \\
& \left. + h_2h_3 + h_2h_4 + 3h_3^2 + 6h_3h_4 + 3h_4^2\right) \qquad (128) \\
\eta =& -\frac{\left(\sum_{i=1}^4 h_i\right)}{60h_4\left(h_3+h_4\right)\left(h_2+h_3+h_4\right)} \left(3h_1^3 + 4h_1^2h_2 - h_1^2h_3 - 6h_1^2h_4 - h_1h_2^2 \right. \\
& \left. - 2h_1h_2h_3 - 2h_1h_2h_4 - h_1h_3^2 + 8h_1h_3h_4 + 9h_1h_4^2 - 2h_2^3 - h_2^2h_3 + 4h_2^2h_4\right.
\end{aligned}$$

$$+4h_2h_3^2 - 2h_2h_3h_4 - 6h_2h_4^2 + 3h_3^3 - 6h_3^2h_4 - 21h_3h_4^2 - 12h_4^3) \tag{129}$$

For equidistant points this simplifies to:

$$\alpha = \frac{14h}{45} = 4h\frac{7}{90} \qquad \beta = \frac{64h}{45} = 4h\frac{32}{90} \qquad \gamma = \frac{8h}{15} = 4h\frac{12}{90}$$
$$\zeta = \frac{64h}{45} = 4h\frac{32}{90} \qquad \eta = \frac{14h}{45} = 4h\frac{7}{90}$$

The expression is $\mathcal{O}\left(h^6\right)$ for non-equidistant points and $\mathcal{O}\left(h^7\right)$ for equidistant points.