Solving the guiding-center model on a regular hexagonal mesh

Michel Mehrenberger, Laura S. Mendoza, Charles Prouveur, Eric Sonnendrücker

Solving the Guiding-Center model on a regular hexagonal mesh

Michel Mehrenberger, Laura S. Mendoza, Charles Prouveur, Eric Sonnendrücker

October 27, 2015

Abstract

This paper introduces a Semi-Lagrangian solver for the Vlasov-Poisson equations on a uniform hexagonal mesh. The latter is composed of equilateral triangles, thus it doesn't contain any singularities, unlike polar meshes. We focus on the guiding-center model, for which we need to develop a Poisson solver for the hexagonal mesh in addition to the Vlasov solver. For the interpolation step of the Semi-Lagrangian scheme, a comparison is made between the use of box-splines and of Hermite finite elements. The code will be adapted to more complex models and geometries in the future.

Introduction

In magnetic fusion applications the embedded closed magnetic flux surfaces play an important role and introduce an important anisotropy[2]. For this reason one gets favorable numerical properties when grid points align on the concentric magnetic flux surfaces. When trying to do this with a mapped cartesian grid, one ends up with a polar coordinates mesh (when the flux surfaces are circles) or something topologically equivalent. This yields smaller and smaller cells when getting closer to the center as well as a singularity at the center. This is numerically far from optimal.

Different strategies have been implemented to avoid these singularities, we can cite among others: the iso-parametric analysis approach done by J. Abiteboul et al. [1] and A. Ratnani [31] or N. Besse and E. Sonnendrücker's work with unstructured meshes[6]. The methods presented in these papers are particularly interesting as not only they avoid singularities but also they are extremely flexible and can be easily adapted to more complex geometries. However, even if these two approaches are different, they suffer from additional cost due to the numerical complexity coming either from the computation of the inverse of a mapping, the advection of the derivatives or the localization of the feet of the characteristics. As for the Poisson equation, a recent study relevant to our problem was made by T. Nguyen et al.[30] to compare different solvers on the disk, where the Iso-geometric approach stood out as a competitive method.

There are three kinds of regular pavings of the plane: using squares, equilateral triangles or hexagons. When considering meshes, the dual mesh of a square mesh, i.e the mesh generated when taking the Voronoi cells of every point of the original lattice, is a shifted square mesh and the regular triangle mesh is the dual of the regular hexagonal mesh (See Figure 1).

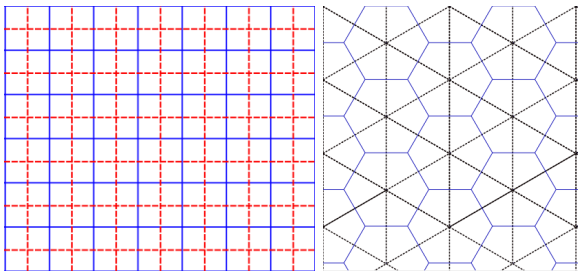

Figure 1: A square mesh and its dual (left), and a hexagonal mesh, in solid blue, and its dual mesh, the triangular tessellation, in dashed black (right).

Tiling a regular hexagon into triangles yields a mesh of equilateral triangles having all the same area. Such a mesh was first introduced for numerical simulations in [32]. An application to particle methods is proposed in [9]. This grid can be easily mapped to a circle by slightly stretching the edges of the hexagon. Indeed, the lattice is actually composed of concentric hexagons, thus the transformation is a simple scaling of the hexagons points to their circumscribed circle. The scaling coefficient is given by the ratio between the radius of circumscribed circle and the distance from the origin to the point. This yields a nice mesh of a disk with slightly stretched triangles of almost the same size and there is no singularity in any point of the domain. Additionally, such a mesh has a structure with three privileged directions, and uniform steps in each direction, thus it is completely straightforward to localise points within this mesh. The derivatives along the three directions can also be nicely computed using the regular finite difference method along the three directions. And last but not least, there is a spline construction on this mesh, called box-spline [13]. These splines have a hexagonal support and are invariant by translations along the three directions of the mesh.

The difficulties mentioned for previous approaches, are not present when dealing with this uniform hexagonal mesh. Regarding the computational efficiency, all our simulations include an analysis and comparison with more common methods. Moreover a simple and non singular mapping from this mesh can be used to handle more complex settings like the surface aligned meshes needed for tokamak simulations. This point will be left for further studies.

In this work, we focus on adapting the Semi-Lagrangian scheme to this hexagonal mesh. This scheme consists basically of two steps: computing the characteristics' origins and interpolating at these points. For the latter, we compare two different approaches: one using box-splines and the second ap-

proach using Hermite Finite Elements. Both interpolation methods, as well as the mesh, are presented in Section 1. In Section 2, we present a simple finite difference Poisson solver adapted to the hexagonal mesh. We introduce a guiding-center approximation of the 2D Vlasov Poisson system[20], and the Semi-Lagrangian scheme to solve it, in Section 3. Finally, in Section 4, we compare the results of the scheme using box-splines with the ones using Hermite finite elements. Besides the guiding-center model, the circular advection model is used to compare the numerical methods.

1 Interpolation on regular hexagonal mesh

1.1 The hexagonal mesh

The hexagonal mesh is obtained by tiling a regular hexagon into equilateral triangles. The mesh obtained can be generated by three vectors. These unit vectors are

$$\mathbf{r_1} = \begin{matrix} \sqrt{3}/2 \\ 1/2 \end{matrix}, \qquad \mathbf{r_2} = \begin{matrix} -\sqrt{3}/2 \\ 1/2 \end{matrix}, \qquad \mathbf{r_3} = \begin{matrix} 0 \\ 1 \end{matrix}. \tag{1}$$

The 2D lattice sites are obtained by the product $\mathbf{Rk}$ where $\mathbf{R} = (\mathbf{r_1}\ \mathbf{r_2})$ and $\mathbf{k} = (k_1, k_2)^T \in \mathbb{Z}$. To obtain exactly the mesh as in Figure 2, we need to define a few extra parameters: an origin, denoted by $P_0(x_0, y_0)$, a radius L which is the distance between the origin and any external vertex of the hexagon and the number of cells N_c on any radius.

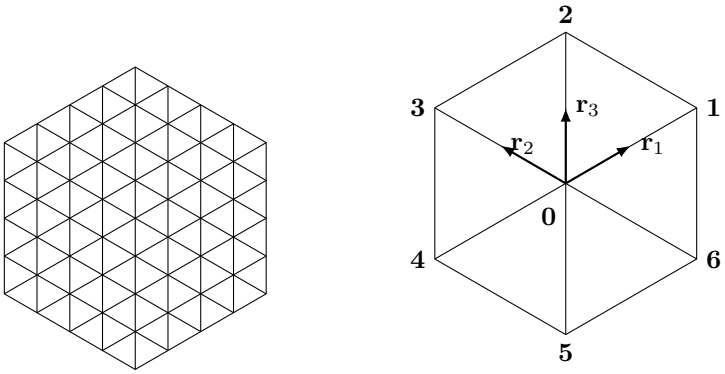

Figure 2: The hexagonal lattice and the vectors $\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}$ that generate such a mesh

The mesh is based on uniform hexagons of the first type (see [35]). For local and global indexing we will use the following convention: the point at the center will be the point of index 0. Following the direction $\mathbf{r_1}$ the next point will be indexed 1, and the indexing will follow in a counter-clockwise motion. And so

on, until all the points of the domain have been indexed. See Figure 2. We will denote H_i the unit hexagon cell that is centered at the point of global index i.

Besides the fact that the hexagonal mesh contains no singularities, its regularity allows us to localize the characteristics' origins for the Semi-Lagrangian scheme by taking three integer values, similarly to what is done on cartesian grids for which only two integer values are needed. Nevertheless, the accuracy of the method depends heavily on the interpolation method chosen. For example, for a Cartesian grid, it is common to use cubic splines which have shown to give accurate results in an efficient manner [33]. In our problem, with the hexagonal lattice, B-splines do not exploit the isotropy of the mesh (for more information see [28]) and are defined by a convolution in 2D, which can't be done for our mesh. Therefore, we need to use another approach. In the following two subsections we present two different strategies: the first one using box-splines and a second one using Hermite Finite Elements.

1.2 Box-splines quasi-interpolation

There are mainly two families of splines that take advantage of the geometry's properties: hex-splines, first introduced in [36], and the three directional box-splines. For a detailed comparison between these two types of splines we will refer to [12]. Based on the latter, we chose to use box-splines, as the results are more stable. And lastly, also based on the previously cited paper, we decided to use a quasi-interpolation method[26].

1.2.1 Box-Splines: General Definition

Box-splines are a generalization of the well known B-splines. They are also piecewise polynomial and they share some properties, such as: compact support, positiveness, symmetry and partition of unity. But, unlike B-splines, box-splines are defined from a generator matrix Ξ. Therefore, to construct them on the hexagonal lattice, we will use the generator vectors $\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}$. The general definition is [16, 11]:

Definition 1.1 (Box-splines) *Let Ξ be a $d \times m$ matrix with non-null columns in $\mathbb{R}^d$. A box-spline χ_Ξ associated to the matrix Ξ, is a multivariate function $\chi_\Xi : \mathbb{R}^d \longrightarrow \mathbb{R}$. If Ξ is a square invertible matrix,* i.e. *when $m = d$ and $\det(\Xi) \neq 0$, we define a box-spline with the formula below*

$$\chi_\Xi(\mathbf{x}) = \begin{cases} \dfrac{1}{|\det(\Xi)|} & \text{if } \Xi^{-1}\mathbf{x} \in [0,1)^2, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

If $\Xi \cup \mathbf{v}$ is a $d \times (m+1)$ matrix, composed by the m column vectors from Ξ to which we append the vector $\mathbf{v}$, we define the box-spline $\chi_{\Xi\cup\mathbf{v}}$ by recursion:

$$\chi_{\Xi\cup\mathbf{v}}(\mathbf{x}) = \int_0^1 \chi_\Xi(\mathbf{x} - t\mathbf{v})\, dt. \tag{3}$$

Remark 1.1 *We notice that uniform B-Splines are box-splines where the generating matrix is* $\Xi = h\,(e_1,\ e_2)$*, where* $e_1 = (0,1)^T$ *,* $e_2 = (1,0)^T$ *and* $h \in \mathbb{R}^+$ *is the step of the uniform mesh. For a B-spline of degree* d*, the multiplicity of* e_1 *and* e_2 *are both* $d+1$*.*

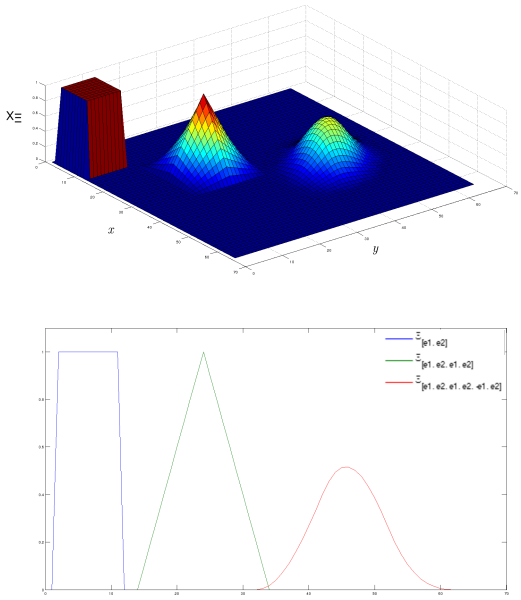

Figure 3: On the left: Box-splines $\chi_{[e_1,e_2]}$, $\chi_{[e_1,e_2,e_3]}$ (shifted for better visualization), and $\chi_{[e_1,e_2,e_3,e_4]}$ (also translated), where $e_1 = (0,1)^T$, $e_2 = (1,0)^T$, $e_3 = e_1 + e_2$, and $e_4 = e_1 - e_2$. On the right: the 2d projection of the box-splines onto the x plane.

Remark 1.2 *The box-splines can have different degrees in each direction. Thus, there are different definitions of the degree. We will adopt the definition below, which will be specific to the kind of box-splines we use in this paper.*

Definition 1.2 (Three-directional Box-spline of degree N**)** *Let* Ξ *be a* 2×3 *matrix with non-null columns in* $\mathbb{R}^2$ *such that the column vectors of* Ξ *form a generating basis of* $\mathbb{R}^2$*. Then, the three-directional box-spline of degree* N *of generating matrix* Ξ*,* χ_Ξ^N*, is the box-spline associated to* Ξ*, where all three generating vectors have multiplicity* N*.*

1.2.2 The quasi-interpolation scheme

The problem is the following: we are given an initial sample on the grid $s[\mathbf{k}] = f_0(\mathbf{Rk})$, where the points $\mathbf{Rk}$ belong to our hexagonal mesh, and we need to

know the values $f(\mathbf{x})$ where $\mathbf{x} \notin \mathbf{Rk}$. To this aim we define a spline surface $f(\mathbf{x}) = \sum c[\mathbf{k}]\chi^N(\mathbf{x} - \mathbf{Rk})$, where χ^N are the box-splines of degree N of matrix $\Xi = [\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}]$ and $c[\mathbf{k}]$ are the coefficients associated to them. These coefficients are defined such that $f(\mathbf{x})$ approximates $f_0(\mathbf{x})$ to a certain order $M = 2N$ or, in other words, the approximation is exact only if $f_0(\mathbf{x})$ is a polynomial of degree $M - 1$ or less [12]. This is different from the classical interpolation method, where the reconstruction is exact on grid points for all smooth functions. The $c[\mathbf{k}]$ coefficients are the box-splines coefficients, to compute them we are no longer able to solve a matrix-vector system because of the extra degree of freedom given by the quasi-interpolation method. Thus, the $c[\mathbf{k}]$ coefficients are obtained, for a grid point $\mathbf{x}_j = \mathbf{Rk}_j$, by discrete filtering[14]

$$c[\mathbf{k_j}] = \sum_{i=0}^{K-1} s[\mathbf{k}_i + \mathbf{k}_j]p[i], \tag{4}$$

where s is the initial sample data and $p[i]$ are K pre-filters which will be defined later on.

Remark 1.3 *For a hexagonal domain, this type of quasi-interpolation is adequate. Nevertheless, we still need to study what would happen when applying the IgA approach (when introducing a domain transformaiton) to the hexagonal mesh.*

1.2.3 Box-splines coefficients

We recall we have formula (4). Based on the literature available, notably [12], we have chosen for second-order box-splines the quasi-interpolation pre-filters p_{IIR2} which seem to give better results within a competitive time. The pre-filter $p_{IIR2}[i]$ of the point of local index i, for splines of degree 1, is defined as follows:

$$p_{IIR2}[i] = \begin{cases} 1775/2304, & \text{if } i = 0, \\ 253/6912, & \text{if } 0 < i < 7, \\ 1/13824, & \text{if } 6 < i < 19 \text{ and } i \text{ odd}, \\ 11/6912, & \text{if } 6 < i < 19 \text{ and } i \text{ even}, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Here $K = 19$, and for box-splines of degree 2, $K = 61$. For higher degrees, we refer to the previously mentioned papers (particularly [14]). To get the Box-splines coefficients, we use (4), where $p[i] = p_{IIR2}[i]$.

1.2.4 Optimizing the evaluation

At the present state we have all the elements for the approximation of a function f with box-splines of degree N

$$\tilde{f}(\mathbf{x}) = \sum_{\mathbf{k} \in \mathbb{Z}^2} c[\mathbf{k}]\chi^N(\mathbf{x} - \mathbf{Rk}). \tag{6}$$

Even if we limit our sum to the vector $\mathbf{k}$ that defines our domain, we would like to take advantage of the fact that the splines χ^N are only non-zeros in a limited number of points. Therefore we need to know the indices $\mathbf{k}$ such that $\chi^N(\mathbf{x}-\mathbf{Rk}) \neq 0$. For this purpose we will use the strategy suggested in [12]: to start we need to obtain the indices on the coordinate system generated by $\mathbf{R}$: $\mathbf{k}_0 = [\lfloor u \rfloor \ \lfloor v \rfloor]$ where $(u\ v)^T = \mathbf{R}^{-1}\mathbf{x}$. Thus, for example, in the case $N = 1$, we only need 4 terms associated to the encapsulating rhomboid's vertices: $\mathbf{Rk}_0$, $\mathbf{Rk}_0 + \mathbf{r_1}$, $\mathbf{Rk}_0 + \mathbf{r_2}$ and $\mathbf{Rk}_0 + \mathbf{r_1} + \mathbf{r_2}$. Finally we obtain:

$$\begin{aligned}\tilde{f}(\mathbf{x}) = \quad & c[\mathbf{k}_0]\ \chi^1(\mathbf{x} - \mathbf{Rk}_0) \\ & + c[\mathbf{k}_0 + [1,0]]\ \chi^1(\mathbf{x} - \mathbf{Rk}_0 - \mathbf{r_1}) \\ & + c[\mathbf{k}_0 + [0,1]]\ \chi^1(\mathbf{x} - \mathbf{Rk}_0 - \mathbf{r_2}) \\ & + c[\mathbf{k}_0 + [1,1]]\ \chi^1(\mathbf{x} - \mathbf{Rk}_0 - \mathbf{r_1} - \mathbf{r_2}). \end{aligned} \tag{7}$$

Remark 1.4 *The χ^1 spline has a support of radius the unity, thus one of the elements of* (7) *is null. But this formula allows us to keep a short general formula for all points on the mesh without having to compute the indices of the cell to which* $\mathbf{x}$ *belongs to.*

Remark 1.5 *For the box-splines of degree* 2*, in equation* (7) *there would be* 16 *coefficients to compute (see Figure 4) from which* 4 *would be null terms.*

1.2.5 Algorithms

Finally, we give the complete algorithms for quasi-interpolating a function f with box-splines, which relies on two further algorithms: an algorithm for computing the quasi-interpolating coefficients, and an algorithm for computing the box-splines.

Algorithm 1: Quasi-interpolation with box-splines

Data: Domain denoted Ω. The function f is known at the mesh points $\mathbf{x}_i$, sample$(i) = f(\mathbf{x}_i)$.
Result: Approximate value of the function f at arbitrary points $\mathbf{x}$ (result)

```
forall the x do
    Initialize result = 0;
    Initialize coeffs = BoxSplineCoeff(sample, degree);
    Compute hexagonal coordinates: k_0 = [⌊u⌋ ⌊v⌋] where (u v)^T = R^{-1} x
    ;
    for k_1 = 1− degree to degree do
        for k_2 = 1− degree to degree do
            /* Treat the points on the enveloping rhomboid of
            radius = degree */
            Compute hexagonal coordinates of point in vicinity of x using
            (k_1, k_2) ⟶ k̂ ;
            if k̂ ∈ Ω then
                Compute global index of k̂ ⟶ index;
                Get cartesian coordinates of point at index ⟶ x̂;
                result = result + coeffs(index) * BoxSplineValue(x̂,
                degree)
            end
        end
    end
end
```

Algorithm 2: Computation of Box-spline coefficients

Data: sample, array containing the values of $f(\mathbf{x}_i)$.
degree, the degree of the splines for quasi-interpolation.
Result: coeffs, array containing the box-splines coefficients at each mesh point $\mathbf{x}_i$

```
Initialize K = number of points in the vicinity of x_i (depends on the type
of pre-filter used);
Initialize PreFilter = array of local pre-filters. ;      /* See (1.2.3) */
forall the x_i = x ∈ Ω do
    Initialize coeffs(i) = 0;
    for local = 1 to K do
        Compute global index of point at local of x ⟶ global;
        coeffs(i) = coeffs(i) + sample(global) * PreFilter(local)
    end
end
```

As for the third and last algorithm, the procedure to compute a box-spline of a certain degree on a point, we refer to [11]. We used the same evaluation for box-splines. Furthermore, we focused on degree 2 splines as they have been

optimized in this same article.

1.3 Hermite Finite Elements interpolation

Another type of interpolation method is the Hermite Finite Element interpolation. In which, to interpolate a function f at a point $\mathbf{x}$ of barycentric coordinates $(\lambda_1, \lambda_2, \lambda_3)$ in the triangle T of vertices S_1, S_2 and S_3, we need a finite element with a local interpolation operator Π_T. This operator can be defined by the duality product of a set of degrees of freedom Σ_T and a set of basis functions B which depend on the barycentric coordinates. For the following section, we define the indices i, j, and k, with the following relations:

$$i \in [\![1;3]\!],\ j = i[3]+1,\ k = j[3]+1.$$

Here $i[3]$ (respectively $j[3]$) is the rest of the euclidean division of i (resp. j) by 3. Using this notation, for any vertex S_i of a cell T, the remaining two vertices are S_j and S_k.

Several elements have been tested here: The Z9 and Z10 Zienkiewicz elements, the Hsieh-Clough-Tocher reduced (HCT-r) and complete (HCT-c), the Ganev-Dimitrov element, and lastly the Mitchell element. These elements can be found in [22], [4], and [29]. We show in the following sections, specifically how the hexagonal structure simplifies the interpolation with these elements. However, before presenting in details each different type of element, we start this section with a brief overview on how we compute the derivatives, which are needed in all methods.

1.3.1 Derivatives reconstruction

In our setting, only values at the vertices are known, whereas other degrees of freedom are reconstructed through finite difference formula, as in [23] for example. We focus here in the reconstruction of derivatives in a given direction.

Given a function $f \in \mathbb{R}^2$ and a direction $\mathbf{h} \in \mathbb{R}^2$, the right derivative (symbolized by the sign +) of order $p \in \mathbb{N}^*$ along the direction $\mathbf{h}$ of f at a point $\mathbf{x}_i \in \mathbb{R}^2$ is denoted by $f'_{\mathbf{h}^+}(\mathbf{x}_i)$ and is approximated by the formula

$$hf'_{\mathbf{h}^+}(\mathbf{x}_i) \simeq \sum_{\ell=r^+}^{s^+} b_\ell^+ f(\mathbf{x}_i + \ell h) \tag{8}$$

where $h = |\mathbf{h}|$, is the euclidian length of $\mathbf{h}$, $r^+ = -\lfloor \frac{p}{2} \rfloor$, $s^+ = \lfloor \frac{p+1}{2} \rfloor$, and

$$\begin{cases} b_\ell^+ = \dfrac{\prod_{\kappa=r^+,\ \kappa\notin\{0,\ell\}}^{s^+}(-\kappa)}{\prod_{\kappa=r^+,\ \kappa\neq\ell}^{s^+}(\ell-\kappa)},\ \text{for } \ell = r^+,\dots,s^+,\ \ell\neq 0, \\ b_0^+ = -\displaystyle\sum_{\kappa=r^+,\ \kappa\neq 0}^{s^+} b_\kappa^+ \end{cases}. \tag{9}$$

Formula (9) yields, for the orders $p = 1 \dots 6$, the following coefficients.

$$\begin{array}{ll} p=1 & b^+ = (-1,\ 1), \\ p=2 & b^+ = (-1/2,\ 0,\ 1/2), \\ p=3 & b^+ = (-1/3,\ -1/2,\ 1,\ -1/6), \\ p=4 & b^+ = (1/12,\ -2/3,\ 0,\ 2/3,\ -1/12), \\ p=5 & b^+ = (1/20,\ -1/2,\ -1/3,\ 1,\ -1/4,\ 1/30), \\ p=6 & b^+ = (-1/60,\ 3/20,\ -3/4,\ 0,\ 3/4,\ -3/20,\ 1/60). \end{array} \tag{10}$$

Using these coefficients in formula (8), we can get the derivatives along each direction $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$, by taking $\mathbf{h} \in \{\pm\mathbf{r}_i,\ i = 1, 2, 3\}$. We notice that this will be equivalent to obtain the derivatives along the edges of any cell of our mesh.

1.3.2 The Z9 and Z10 Zienkiewicz elements

The Z9 approach[22] uses 9 degrees of freedom which are the values of f at the vertices of the triangle (which are given) and the values of the derivatives in the direction of the edges at every vertex (computed with the procedure explained in the previous section 1.3, Formula (8), with $\partial_{\overrightarrow{S_iS_j}} f(S_i) = f'_{\mathbf{h}^+}(S_i)$ and $\mathbf{h} \in \{\pm\mathbf{r}_i,\ i = 1, 2, 3\}$),

$$\Sigma_T = \{\forall i \in [\![1; 3]\!], f(S_i), \partial_{\overrightarrow{S_iS_j}} f(S_i), \partial_{\overrightarrow{S_iS_k}} f(S_i)\}. \tag{11}$$

Z10 uses one more degree of freedom which is the value at C, the center of the triangle:

$$\Sigma_T = \{\forall i \in [\![1; 3]\!], f(S_i), \partial_{\overrightarrow{S_iS_j}} f(S_i), \partial_{\overrightarrow{S_iS_k}} f(S_i), f(C)\}. \tag{12}$$

The advantage of using the Z10 element is the gain of one order of precision: it reproduces polynomials of total degree ≤ 3 whereas with Z9 only polynomials of degree ≤ 2 are reproduced. Let us note that although adding one degree of freedom seems harmless, adding the cells' centers represents, for a hexagonal mesh, a computational cost three times higher. In fact, there are twice as many centers as vertices in a hexagonal mesh. Therefore the number of computational points is tripled.

Let us define the basis functions needed to interpolate with the element Z9. First denote by

$$\phi = \lambda_1\lambda_2\lambda_3. \tag{13}$$

We introduce now ξ_i and ξ_{ij} for performances purposes:

$$\xi_i = \lambda_i^3 - \phi \quad \text{and} \quad \xi_{ij} = \lambda_i^2\lambda_j + \frac{\phi}{2}.$$

And thus we find that the basis functions associated to the vertex and derivative degrees of freedom write respectively

$$\phi_i = 3\lambda_i^2 - 2\xi_i \quad \text{and} \quad \phi_{ij} = h_{ij}\xi_{ij} = h\xi_{ij},$$

where h_{ij} is the length of $[S_iS_j]$, and ϕ_i is the basis function associated with the value of the function at S_i while ϕ_{ij} are associated with the derivatives in the direction of the edges. The fact that T is equilateral is exploited here by replacing h_{ij} with h since the length of $[S_iS_j]$ is constant. Finally for Z9 we have:

$$\Pi_T(f) = \sum_{i=1}^{3} [f(S_i)\,\phi_i + \sum_{j\neq i} \partial_{\overrightarrow{S_iS_j}} f(S_i)\,\phi_{ij}]. \tag{14}$$

In the same manner, let us define the basis functions needed to interpolate with Z10: Firstly, we introduce ϕ_i the basis function at the vertices,

$$\phi_i = 3\lambda_i^2 - 2\xi_i - 9\phi.$$

Secondly, ϕ_{ij}, the basis functions derivatives along the edges,

$$\phi_{ij} = h_{ij}(\xi_{ij} - \frac{3}{2}\phi) = h(\xi_{ij} - \frac{3}{2}\phi).$$

And lastly, ϕ_{123}, at the cell's center,

$$\phi_{123} = 27\phi.$$

Thus, for $Z10$ we have:

$$\Pi_T(f) = \sum_{i=1}^{3} [f(S_i)\,\phi_i + \sum_{j\neq i} \partial_{\overrightarrow{S_iS_j}} f(S_i)\,\phi_{ij}] + f(C)\,\phi_{123}. \tag{15}$$

1.3.3 The HCT elements

The HCT elements were tested as well because of their original feature which is to use a division of the triangle into three sub-triangles. This characteristic is the only difference between the interpolation with the HCT-r and the Z9 element as they both use the same 9 degrees of freedom, and reproduce polynomials of degree ≤ 2. Unsurprisingly, they give quasi-identical results which is why we will not detail the HCT-r interpolation and focus solely on the HCT-c elements. These elements use the same degrees of freedom as HCT-r, the values at the vertices and their derivatives along the edges, but it has additionally the values of the derivatives in the normal direction of the edges at the middle of the respective edge. This adds up to twelve degrees of freedom. The HCT-c elements reproduce polynomials of degree ≤ 3. For a detailed description of these elements on an unstructured mesh we refer to [5], here we focus on applying these schemes to our mesh whose regularity simplifies greatly the computation of the derivatives (see 1.3.1).

Let us now define its interpolation operator. Let S_i be a vertex of the triangle T, then we define respectively l_i and m_i as the length and the middle of the edge opposite to S_i. The exterior normal to the edge opposite of S_i is noted ν_i. Let G be the barycenter of T, then K_l is the sub-triangle made with G, S_j and S_k, we get

$$\Pi_{K_l}(f) = \sum_{i=l}^{(l+1)\mathrm{mod}(3)+1} \sum_{\substack{j=l \\ j\neq i}}^{(l+1)\mathrm{mod}(3)+1} [f(S_i)\,\phi_{l,i} + \partial_{\overrightarrow{S_iS_j}} f(S_i)\,\phi_{l,ij} - n_i \partial_{\nu_i} f(m_i)\phi^{\perp}_{l,i}], \tag{16}$$

where $n_i = |m_i - S_i|$, and $\phi_{l,i}$ is the basis function in the sub-triangle K_l at the vertex S_i, $\phi_{l,ij}$ the first derivative of $\phi_{l,i}$ along the edge S_iS_j and finally, $\phi^{\perp}_{l,i}$ is the derivative along the normal passing through S_i to the opposite edge. For computing the derivatives along the edges, we use formula (8), with $h\partial_{\overrightarrow{S_iS_j}} f(S_i) = h f'_{\mathbf{h}^+}(S_i)$ and $\mathbf{h} \in \{\pm \mathbf{r}_i,\ i = 1, 2, 3\}$. For computing the normal derivatives at m_i, please see Remark 1.6. The basis functions are defined by

$$B_l = \Sigma_l \Lambda_l, \tag{17}$$

with Σ_l the matrix containing the basis functions' coefficients in the sub-triangle K_l. Furthermore,

$$B_l = (\phi_{l,i},\ \phi_{l,j},\ \phi_{l,k},\ \phi_{l,ik},\ \phi_{l,ij},\ \phi_{l,ji},\ \phi_{l,jk},\ \phi_{l,kj},\ \phi_{l,ki},\ \phi^{\perp}_{l,i},\ \phi^{\perp}_{l,j},\ \phi^{\perp}_{l,k})^T,$$

and

$$\Lambda_l = (\lambda_i^3,\ \lambda_j^3,\ \lambda_k^3,\ \lambda_i^2\lambda_k,\ \lambda_i^2\lambda_j,\ \lambda_j^2\lambda_i,\ \lambda_j^2\lambda_k,\ \lambda_k^2\lambda_j,\ \lambda_k^2\lambda_i,\ \lambda_i\lambda_j\lambda_k)^T.$$

The matrices Σ_l are defined with the eccentricity e_i of each edge opposite to the vertex S_i of the triangle T:

$e_i = \dfrac{l_k^2 - l_j^2}{l_i^2}$ where j and k are the indices of the remaining two vertices of T – S_j and S_k.

For an equilateral triangle, the eccentricity is null which simplifies greatly Σ_l:

$$\Sigma_l = \begin{pmatrix}
0 & 0 & 0 & \frac{9}{2} & \frac{9}{2} & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{2} & 1 & 0 & \frac{-3}{2} & 0 & 3 & 3 & 0 & 0 & 3 \\
\frac{1}{2} & 0 & 1 & 0 & \frac{-3}{2} & 0 & 0 & 3 & 3 & 3 \\
\frac{-1}{12} & 0 & 0 & \frac{5}{4} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{12} & 0 & 0 & \frac{1}{2} & \frac{5}{4} & 0 & 0 & 0 & 0 & 0 \\
\frac{-7}{12} & 0 & 0 & \frac{-1}{2} & \frac{-1}{4} & 1 & 0 & 0 & 0 & 1 \\
\frac{2}{3} & 0 & 0 & \frac{-1}{4} & \frac{1}{4} & 0 & 1 & 0 & 0 & \frac{1}{2} \\
\frac{2}{3} & 0 & 0 & \frac{1}{4} & \frac{-1}{4} & 0 & 0 & 1 & 0 & \frac{1}{2} \\
\frac{-7}{12} & 0 & 0 & \frac{1}{4} & \frac{-1}{4} & 0 & 0 & 1 & 0 & \frac{1}{2} \\
\frac{2}{3} & 0 & 0 & \frac{1}{4} & \frac{-1}{4} & 0 & 0 & 1 & 0 & \frac{1}{2} \\
\frac{-7}{12} & 0 & 0 & \frac{1}{4} & \frac{-1}{4} & 0 & 0 & 1 & 0 & \frac{1}{2} \\
\frac{1}{4} & 0 & 0 & \frac{-1}{4} & \frac{-1}{2} & 0 & 0 & 0 & 1 & 1
\end{pmatrix}. \tag{18}$$

The advantage of HCT-c and HCT-r is that they don't require more points than the lattices of the hexagonal mesh. This is not the case for the Z10 approach.

Remark 1.6 *The normal derivatives at the middle of the edges are computed using the derivatives following the same normal vector but at the vertices of the triangle, which are themselves computed using the derivatives at the vertices but with respect to the edges (i.e.* $\partial_{\overrightarrow{S_iS_j}} f(S_i)$*). For example, let* $\partial_\nu f(m)$ *be the normal derivative at the middle of the edge* $[S_iS_j]$*. We first compute all* $\partial_\nu f(S_i + \kappa \overrightarrow{S_iS_j})$*, where* $\kappa = -2, \ldots, 3$ *using a linear combination of* $\partial_{\mathbf{h}} f(S_i)$ *with* $\mathbf{h} \in \{\mathbf{r}_1, \mathbf{r}_2\}$*. To obtain the normal derivatives at the middle of the edges, we simply use a Lagrange interpolation of degree* 5*, such as:*

$$\partial_\nu f(m) = \sum_{\kappa=-2}^{3} a_\kappa \partial_\nu f(S_i + \kappa \overrightarrow{S_iS_j}),$$

where

$$a_{-2} = a_3 = \frac{3}{256}, \quad a_{-1} = a_2 = \frac{-25}{256}, \quad \text{and } a_0 = a_1 = \frac{75}{128}.$$

1.3.4 The Ganev-Dimitrov element

The Ganev-Dimitrov element reproduces polynomials of total degree ≤ 4 and uses 15 degrees of freedom which are the values of the function at the vertices and at the middle of the edges, plus the value of the derivatives at the vertices in the direction of the other two vertices. The computational cost for this element is four times higher than the HCT-r interpolation because of the computations needed at the middle of the edges: there are on average 3 times more edges than vertices. As a matter of fact, the vertices and the middle of the edges form another hexagonal mesh twice as fine as the original mesh. The reason why we tested such a computationally expensive element is to observe whether or not the gain in precision is interesting compared to the extra computing time allocated. The local interpolation operator is:

$$\Pi_T(f) = \sum_{i=1}^{3} [f(S_i)\,\phi_i + \partial_{\overrightarrow{S_iS_j}} f(S_i)\,\phi_{ij} + f(m_i)\,\phi_i^{\perp,0} - n_i \partial_{\nu_i} f(m_i) \phi_i^{\perp,1}]. \quad (19)$$

Contrary to (16), here we use the values of normal derivatives of the basis function at the middle point of the edge opposite to S_i, $\phi_i^{\perp,1}$, but also the value at these middle points $\phi_i^{\perp,1}$. The basis function are defined by:

$$B = \Sigma\Lambda, \quad (20)$$

with:

$B = (\phi_1,\ \phi_2,\ \phi_3,\ \phi_{1,3},\ \phi_{1,2},\ \phi_{2,1},\ \phi_{2,3},\ \phi_{3,2},\ \phi_{3,1},\ \phi_1^{\perp,0},\ \phi_2^{\perp,0},\ \phi_3^{\perp,0},\ \phi_1^{\perp,1},\ \phi_2^{\perp,1},\ \phi_3^{\perp,1})^T,$

$\Lambda = (\lambda_1^4,\ \lambda_2^4,\ \lambda_3^4,\ \lambda_1^3\lambda_3,\ \lambda_1^3\lambda_2,\ \lambda_2^3\lambda_1,\ \lambda_2^3\lambda_3,\ \lambda_3^3\lambda_2,\ \lambda_3^3\lambda_1,\ \lambda_2^2\lambda_3^2,\ \lambda_3^2\lambda_1^2,\ \lambda_1^2\lambda_2^2,\ \lambda_1^2\lambda_2\lambda_3,\ \lambda_1\lambda_2^2\lambda_3,\ \lambda_1\lambda_2\lambda_3^2),$

and,

$$\Sigma = \begin{pmatrix}
1 & 0 & 0 & 4 & 4 & 0 & 0 & 0 & 0 & 0 & -5 & -5 & -4 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 4 & 4 & 0 & 0 & -5 & 0 & -5 & 0 & -4 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 4 & 4 & -5 & -5 & 0 & 0 & 0 & -4 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -0.5 & -0.5 & 0.5 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -0.5 & -0.5 & 0.5 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & -0.5 & -0.5 & 0.5 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & -0.5 & -0.5 & 0.5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & -0.5 & 0.5 & -0.5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0.5 & -0.5 & -0.5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 0 & 0 & -16 & 16 & 16 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 0 & 16 & -16 & 16 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 16 & 16 & -16 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 4 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & -4 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 4 & -4
\end{pmatrix}$$

Remark 1.7 *Here the normal derivatives are computed in the same matter as for the HCT elements. Nevertheless, this could probably be improved for the Ganev-Dimitrov elements as we have an additional information: the value of the function at the middle of the edges.*

1.3.5 The Mitchell finite element [29] MI(p)

The Mitchell elements use 12 degrees of freedom which are the values of the function at the vertices, the value of the derivatives at the vertices in the direction of the other two vertices, and additionally, it uses mixed derivatives. We will first define the complete Mitchell which uses 15 degrees of freedom, by adding the value of the fourth derivative at the middle of the edges, and then we will set these degree of freedom to zero, as we do not want to compute numerically fourth order derivatives. Given a function $f : \mathbb{R}^2 \to \mathbb{R}$ and a triangle S_1, S_2, S_3, with vertex $S_i = (x_i, y_i) \in \mathbb{R}^2$, we will define the degrees of freedom through the following auxiliary function $g(\lambda_1, \lambda_2, \lambda_3) = f(\lambda_1 S_1 + \lambda_2 S_2 + \lambda_3 S_3)$, which takes as input the barycentric coordinates.
The first 3 degrees of freedom are the values at the vertices of the triangle:

$$g_0 = g(1, 0, 0),\ g_1 = g(0, 1, 0),\ g_2 = g(0, 0, 1).$$

Note that g_1 and g_2 are obtained cyclically from g_0. The following degrees of freedom are the derivatives at the vertices

$$g_3 = h\partial_{\overrightarrow{S_1S_2}} f(S_1) = \lim_{t\to 0^+} \frac{g(1-t, t, 0) - g(1, 0, 0)}{t},\ g_4 = h\partial_{\overrightarrow{S_2S_1}} f(S_2) = \lim_{t\to 0^+} \frac{g(t, 1-t, 0) - g(0, 1, 0)}{t}$$

and

$$g_5 = h^{(4)}(1/2),\ h(t) = g(t, 1-t, 0) = 0.$$

We get the next 6 degrees of freedom cyclically. Mixed derivatives are obtained in the following way

$$g_{12} = h^2 \partial_{\overrightarrow{S_1S_3}} \partial_{\overrightarrow{S_1S_2}} f = \lim_{s,t \to 0^+} \frac{g(1-t-s,t,s) - g(1-t,t,0) - g(1-s,0,s) + g(1,0,0)}{st},$$

and cyclically g_{13} and g_{14}. Numerically, they are obtained by applying the derivation procedure (8) to previously computed derivatives (i.e. $\partial_{\overrightarrow{S_iS_j}} f(S_i)$).

Then given $g_i,\ i = 0, \ldots, 14$, there exists a unique polynomial of degree ≤ 4 that has these degrees of freedom, and this defines the complete Mitchell finite element. The basis functions can be explicitly computed as follows.
We define

$$\begin{aligned}
b_0(\lambda_1,\lambda_2,\lambda_3) &= \lambda_1^2(3 - 2\lambda_1 + 6\lambda_2\lambda_3), \\
b_1(\lambda_1,\lambda_2,\lambda_3) &= \lambda_1^2\lambda_2(1 + 2\lambda_3), \\
b_2(\lambda_1,\lambda_2,\lambda_3) &= b_1(\lambda_2,\lambda_1,\lambda_3), \\
b_3(\lambda_1,\lambda_2,\lambda_3) &= \lambda_1^2\lambda_2\lambda_3, \\
b_4(\lambda_1,\lambda_2,\lambda_3) &= \lambda_1^2\lambda_2^2 \,/\, 24, \\
\sigma(\lambda_1,\lambda_2,\lambda_3) &= (\lambda_2,\lambda_3,\lambda_1)
\end{aligned}$$

and

$$\Pi_T(f) = \sum_{j=0}^{2} (g_j b_0 + g_{3+3j} b_1 + g_{4+3j} b_2 + g_{5+3j} b_4 + g_{12+j} b_3) \circ \sigma^j.$$

This reconstruction is exact for all polynomials of degree ≤ 4. We restrict then this space, setting $g_5 = g_8 = g_{11} = 0$. Thus $\Sigma_T = \{g_i, i = 0 \ldots 14\} \setminus \{g_5, g_8, g_{11}\}$. This corresponds to the space of polynomials of degree ≤ 4 which are of degree ≤ 3 on the edges. As in the previous sections, the derivatives are computed numerically from the values at the points, following the derivation procedure (8). Mixed derivatives are obtained by applying the derivation procedure (8) to previously computed derivatives (i.e. $\partial_{\overrightarrow{S_iS_j}} f(S_i)$).

Note that this interpolation procedure is here the analog on triangles of Hermite interpolation on squares as in [23]. It shares also the same finite element space as the box-splines of degree 2. All this motivates the introduction of this element (which came to our knowledge during the writing of this paper).

In the sequel, we will refer to MI(p), for the Mitchell finite elements using a finite difference reconstruction of order $p \in \mathbb{N}^*$ for the computation of the derivatives and mixed derivatives, using the procedure defined in section 1.3.1 (e.g. MI4, MI6, MI17). We introduce this notation, as unlike for the preceding elements, we will make a study with respect to p (which was set to 6 for the other elements).

2 The Poisson finite-difference solver

When computing the origins of the characteristics with the semi-Lagrangian method for the Vlasov-Poisson or guiding-center models we need to compute the solution of the Poisson equation

$$-\Delta\phi = \rho,$$

ϕ being the potential and ρ the density. We impose here null Dirichlet boundary conditions. In order to solve this equation, we use a simple finite difference scheme. Since the mesh here is hexagonal, a seven point stencil is used as shown in Figure 2. It is composed of the six vertices of a hexagon plus its center. To compute ϕ_0, the value of ϕ at the center 0, the remaining vertices of the hexagon are used. This particular stencil has the property to give a fourth order scheme at little cost [8]. Here is the previously described scheme:

$$-(\phi_1+\phi_2+\phi_3+\phi_4+\phi_5+\phi_6-6\phi_0) = \frac{3h^2}{4}\rho_0 + \frac{h^2}{24}(\rho_1+\rho_2+\rho_3+\rho_4+\rho_5+\rho_6).$$

Compared to the second order scheme on the same stencil, we notice the only difference to be the second term of the equality:

$$-(\phi_1+\phi_2+\phi_3+\phi_4+\phi_5+\phi_6-6\phi_0) = h^2\rho_0.$$

Considering the gain of two order of precision at such little cost, we have used this fourth order scheme to compute ϕ.

Remark 2.1 *One difficulty that arises here is to define an indexing that allows the resolution of a "computational-friendly" linear system, i.e. a sparse matrix with the non-null terms close to the diagonal to minimize filling in a Cholesky decomposition. This is done by assigning a number following one hexagonal direction, row after row, similarly to how one proceeds on a Cartesian mesh. Here, however, the difference is that the rows are of variable width resulting in a banded matrix. Therefore the matrix here is not constituted of 7 diagonals which makes the Poisson computation longer than on a Cartesian mesh. The width of the band is directly proportional to the number of cells in the hexagonal domain.*

3 The Backward Semi-Lagrangian Scheme

When solving a Vlasov equation, one usually thinks of Lagrangian methods such as PIC[7]. However these schemes are prone to numerical noise and converge slowly in $1/\sqrt{N}$ as the number N of particles increases, typical of a Monte Carlo integration. Another option to solve the Vlasov equation, are Eulerian methods like Finite Difference, Finite Element or Finite Volume methods [18, 37, 3]. The downside of this type of method is that there is a numerical limit on the time step, the CFL condition.

With the intent of overcoming the pitfalls of these methods, the Semi-Lagrangian method was introduced, first in numerical weather prediction (see [24] and articles cited within it), and then for plasma simulations [33, 10] and is used also for gyrokinetic simulations of plasma turbulence [21, 25]. This scheme consists in fixing a Eulerian grid in phase-space and following the trajectory of the equation's characteristics in time to compute the evolution of the distribution function. The advantages of this scheme are the possibility of taking large time steps and its stability. However it is still quite costly in high dimensions (5 or 6D phase space) where the PIC method still largely dominates. Lastly, we can point out that there are many types of Semi-Lagrangian solvers (*e.g.* depending on the trajectories: Backward or Forward; depending on degrees of freedom on which it is based: grid points, cell average, ...). We have chosen here to use the classical Backward Semi-Lagrangian (BSL) method.

3.1 Our model

We consider here a 2D linear or non linear advection equation, with a divergence free advection field $\mathbf{A}$, which can be written in general form

$$\frac{\partial \rho}{\partial t} + \mathbf{A} \cdot \nabla_{\mathbf{x}} \rho(\mathbf{x}, t) = 0, \tag{21}$$

where $\mathbf{A}$ is divergence free (*i.e.* $\nabla \cdot \mathbf{A} = 0$) and the density ρ is known at the initial time (*i.e.* $\rho(\mathbf{x}, 0) = \rho_0(\mathbf{x})$ is known). The advection field $\mathbf{A}$ will either be given and known for all times or it will depend on an electric potential computed from the solution of a Poisson equation, *i.e.*

$$\mathbf{A} = \begin{pmatrix} -\frac{\partial \phi}{\partial y} \\ \frac{\partial \phi}{\partial x} \end{pmatrix}, \quad \text{with} - \Delta \phi = \rho.$$

When the advection coefficient is obtained from a Poisson equation, the model is known as the guiding-center model. We will apply the backward Semi-Lagrangian scheme to solve both the advection in a given field and the guiding-center model.

3.2 Computing the origin of the characteristics

We consider the model (21) on a 2D hexagonal domain, discretized with the hexagonal mesh. The points of the lattice are denoted $\mathbf{x} = (x, y)$. The distribution function $\rho(\mathbf{x}, t)$ is known on all grid points at the initial time $t = 0$. Let A_x and A_y be respectively the first and second components of $\mathbf{A}$. We proceed to apply the BSL method to the Vlasov equation (21): First, we need to compute the origin of the characteristics ending at the grid points. These are defined for a given time $s \in \mathbb{R}$ by

$$\left\{\begin{array}{rcl}\dfrac{d\mathbf{X}}{dt} &=& \mathbf{A}\\ \mathbf{X}(s) &=& \mathbf{x}\end{array}\right. \iff \left\{\begin{array}{rcl}\dfrac{dX_1}{dt} &=& A_x\\ \dfrac{dX_2}{dt} &=& A_y\\ X_1(s) &=& x, \quad X_2(s) = y.\end{array}\right. \tag{22}$$

The solutions (X_1, X_2) of (22) are called the characteristics associated to the Vlasov equation. Now denoting by $t^n = n\Delta t$, for a given time step Δt, and $\mathbf{X}^n = \mathbf{X}(t^n)$ for any n, and setting $s = t^{n+1}$. The origin, at time t^n, $\mathbf{X}^n$ of the characteristics ending at the grid point $\mathbf{X}^{n+1} = \mathbf{x}$ can then be computed by any ODE solver, typically a Runge-Kutta solver if $\mathbf{A}$ is known for all times. In the case of the guiding-center model, we use a second order scheme, which is the implicit Adams-Moulton scheme of order two[17], to compute the origin of the characteristics,

$$\frac{\mathbf{X}^{n+1} - \mathbf{X}^n}{\Delta t} = \frac{1}{2}\,(\mathbf{A}^{n+1} + \mathbf{A}^n).$$

Where $\mathbf{A}^n = \mathbf{A}(t^n, \mathbf{X}^n)$. The difficulty here is that $\mathbf{A}(t^{n+1}, \mathbf{X}^{n+1})$, depends on ρ^{n+1} and is unknown, thus an approximation $\overset{*}{\mathbf{A}}$ of $\mathbf{A}$ at time t^{n+1} is made thanks to previous computations:

$$\overset{*}{\mathbf{A}} = 2\ \mathbf{A}(t^n, \mathbf{X}^{n+1}) - \mathbf{A}(t^{n-1}, \mathbf{X}^{n+1}).$$

The unknown $\mathbf{X}^n$ is found by solving:

$$\begin{cases}\dfrac{\mathbf{X}^{n+1} - \mathbf{X}^n}{\Delta t} = \dfrac{1}{2}\,(\overset{*}{\mathbf{A}} + \mathbf{A}^n),\\ \mathbf{X}(s) = \mathbf{x}.\end{cases}$$

Remark 3.1 *Since we need $\boldsymbol{A}(t^{n-1})$, the first step is done using the implicit Euler time scheme.*

3.3 Updating the distribution function

We know that the density ρ is conserved along these characteristics and thus we can write for any time t:

$$\rho(\mathbf{X}(t), t) = \rho(\mathbf{X}(s), s) = \rho(\mathbf{x}, s). \tag{23}$$

So in our case, knowing the origin $\mathbf{X}^n$ of the characteristics, the new value of ρ at t^{n+1} is given by

$$\rho^{n+1}(\mathbf{x}) = \rho^{n+1}(\mathbf{X}^{n+1}) = \rho^n(\mathbf{X}^n), \tag{24}$$

where ρ^n is the distribution function at time step t^n.

The distribution function ρ^n is only known on the mesh points, and the origins of the characteristics $\mathbf{X}^n$ are in general not on a mesh point (see Figure 4). Therefore, we need an interpolation method to compute ρ^n at the characteristic's origin, *i.e.* to approximate $\rho^n(\mathbf{X}^n)$ needed in the equation (24) to get the new value $\rho^{n+1}(\mathbf{x})$ at the grid points, using the known data on the mesh points at its vicinity. This interpolation method will be either: the quasi-interpolation method using Box-splines, where the box-splines coefficients, defined in (4), are computed knowing $s[\mathbf{k}_i] = \rho^n(\mathbf{x}_i)$ (see Section 1.2); or the Hermite Finite-Elements interpolation defined in Section 1.3, where the values at the nodes are used and the derivatives or the values at the middle of the edges are computed by finite difference from the values at the nodes.

3.4 Localizing the characteristics' origins

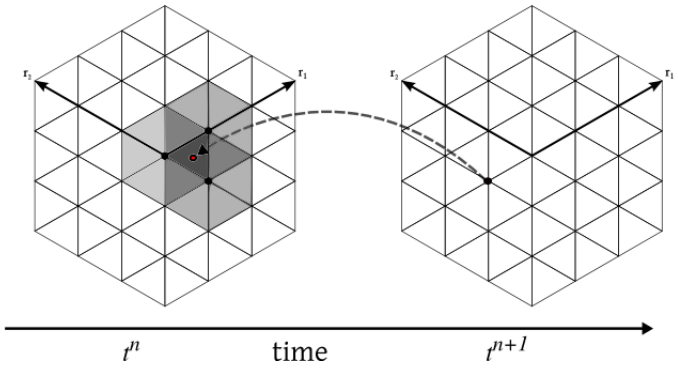

Figure 4: Semi-Lagrangian step: Tracing back characteristics.

One of the advantages of the hexagonal mesh is that it is a uniform mesh. Indeed, even if the mesh is not Cartesian, localizing the characteristics' origin is computationally very efficient, unlike the case of unstructured meshes where iterations are generally required. The procedure is as follows. Let (X_1, X_2) the Cartesian coordinates of the characteristics' origin, obtained by solving (22). Then to obtain the hexagonal coordinates (k_1, k_2) of the lowest point of the rhomboid encapsulating the point, we simply need to solve the system $\mathbf{x} = \mathbf{R}\mathbf{k}$, where $\mathbf{R}$ is the matrix whose columns are the unit vectors given in (1), and take the integer value. Denoting by (r_{ij}) the coefficients of the matrix $\mathbf{R}$, we get

$$\left\{ \begin{array}{rcccl} k_1 & = & \left\lfloor \dfrac{r_{22}X_1 - r_{12}X_2}{r_{11}r_{22} - r_{12}r_{21}} \right\rfloor & = & \left\lfloor \dfrac{1}{\sqrt{3}} (X_1 + X_2) \right\rfloor, \\ k_2 & = & \left\lfloor \dfrac{-r_{21}X_1 + r_{11}X_2}{r_{11}r_{22} - r_{12}r_{21}} \right\rfloor & = & \lfloor X_1 + X_2 \rfloor . \end{array} \right. \tag{25}$$

After obtaining (k_1, k_2), we know the rhomboid (composed by two triangular cells) containing the characteristics' origin. To determine the exact cell on which

the origin is located, we only need to verify if the abscissa of the point is greater than the abscissa of the mesh point at (k_1, k_2) or not. In the first case the point belongs to the cell on the right, else to the cell on the left.

3.5 General algorithm

Below, we summarize the full algorithm to compute the distribution function ρ^{n+1} solution of the guiding-center model (21).

Initialization At time $t = 0$, we suppose that $\rho(\mathbf{x}, 0)$ is given and we evaluate it at the grid points. We denote this data ρ_h^0 (meaning ρ discretized, at the time $t_n = 0$, at the initial step).

Time Loop Incrementation of a given time step Δt, such that: $t^{n+1} = t^n + \Delta t$.

- Solve the Poisson equation to compute advection field $\mathbf{A}^n$;
- Compute the characteristics' origins, $\mathbf{X}^n$, using an ODE solver for (22), Runge-Kutta or Adams-Moulton as described above;
- Interpolate (using either Box-Splines or one of the Hermite Finite Elements) the distribution function ρ^n on $\mathbf{X}^n$ to compute ρ_h^{n+1};
- Update the known values: $\rho^n = \rho^{n+1}$.

Remark Boundary conditions will need to be used between the first and the second step of the time loop (*i.e.* before the interpolation step) for characteristics that leave the computational domain. In this paper we focus only on null Dirichlet boundary conditions.

4 Numerical results

In this section we present the numerical simulations we performed to test our methods. With the aim of studying the convergence, the dissipation, and the efficiency of the schemes, we first study the circular advection test case. To study the accuracy of the results, we compare them to the analytical solution, which is known. Then we proceed to the guiding-center simulation. As there is no analytical solution for this test case, we study quantities of the system that we know should be conserved.

4.1 Circular advection

We focus here on the circular advection test case. The model is defined by:

$$\partial_t f(x, y, t) + y\, \partial_x f(x, y, t) - x\, \partial_y f(x, y, t) = 0. \tag{26}$$

Since this equation is not coupled to a Poisson model, we can study in detail the differences between the interpolation methods previously presented. Additionally, we can find an analytical solution with the method of characteristics

thanks to which we can study the convergence of our schemes. Here, we take a Gaussian pulse as initial distribution function:

$$f_0(x,y) = \exp\left(-\frac{1}{2}\left(\frac{(x-x_c)^2}{\sigma_x^2} + \frac{(y-y_c)^2}{\sigma_y^2}\right)\right), \tag{27}$$

On a hexagonal mesh centered at the origin of radius 8, we take $\sigma_x = \sigma_y = \frac{1}{2\sqrt{2}}$. Let us set here $x_c = 2$ and $y_c = 2$. The distance from the pulse to the limit of the domain makes the boundary effects insignificant, thus we can take a null Dirichlet boundary condition, meaning that we consider that there is no inflow to the domain. To study the convergence in space we took $N_c = 20, 40, 60, ..., 160$. We recall that N_c is the number of cells on the radius L. With the maximum time of evaluation, t_{max}, at 6π, we chose to keep a constant CFL (here there is not an actual CFL, but we decide to keep the ratio $\Delta t/\Delta x$ constant).

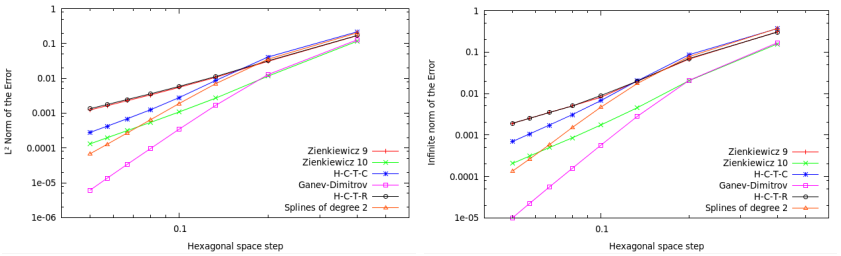

Figure 5: Order of convergence with CFL = 2, with L_2 norm (left) and L_∞ norm (right)

In Figure 5, we plotted the L_2 and L_∞ norms for different space discretization. We can see that for coarse meshes, all the methods are globally the same, with a slightly better accuracy for elements Z10 and Ganev-Dimitrov. But as the mesh gets finer, we can quickly see that the splines converge quicker to better results. Only the Ganev-Dimitrov elements are more accurate.

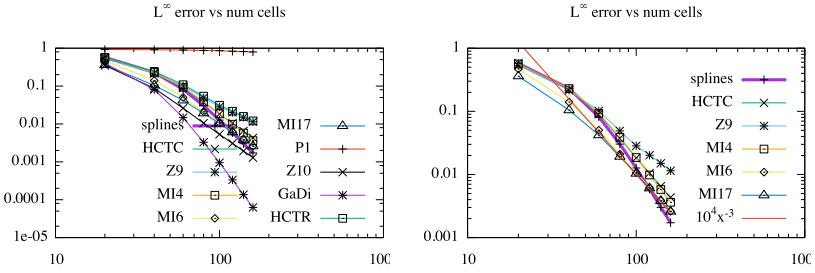

Figure 6: Order of convergence (inclusion of Mitchell elements; CFL = 0.25) of all the methods (left) and only for schemes of order 3 (right).

In Figure 6 (left), we represent the error in L^∞ norm versus the number of

cells, N_c. The figure is similar to the previous one (see Figure 5), but we added the Mitchell elements for comparisons and changed the CFL to CFL = 0.25. The Ganev-Dimitrov element (GaDi) leads again to the best result and the second is Z10. Both have more degrees of freedom (4 times more for GaDi and 3 times more for Z10) for a fixed number of cells N_c. Others reconstructions have the same number of degrees of freedom. Furthermore, we notice that the $P1$ element (classical linear interpolation) leads to very poor accuracy, which fully justifies the use of higher order methods.

In Figure 6 (right), we selected the order 3 schemes (although we added also Z9, which is not exactly accurate to the order three), and added the third order slope for comparison. On the one hand, we observe a super-convergence property for the splines; this may be due to the test-case (i.e. due to its symmetry) or to the quasi-interpolation that is, in some cases, more accurate than a classical interpolation[14]. On the other hand, the Mitchell elements behave favorably: MI4 is as accurate as HCTC and MI6, MI17 are the most accurate, before the splines take over due to this super-convergence, whereas MI6 and MI17 remain third order. We expect higher order convergence when the CFL goes to zero; see [23]). The Z9 method is the least accurate, as expected. Note also that with respect to Figure 5, the error is bigger for all the methods; being that the number of interpolations done is multiplied by 8, as we go from CFL = 2 to 0.25.

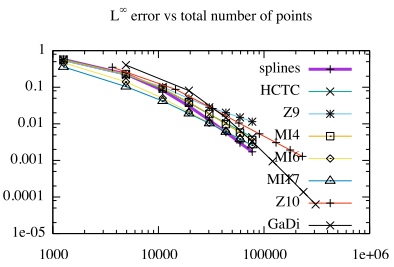

Figure 7: Order of convergence (CFL = 0.25), with respect of number of points

All previous figures study the convergence with respect to the hexagonal step or the number of cells N_c, thus in Figure 7 we wanted to use the total number of points instead. We can see that the most noticeable difference is that the scheme using Ganev-Dimitrov elements is the worst at the beginning whereas in Figure 6 it is the best one since the beginning. This is probably due to the approximation of the normal derivatives, as we noted in Remark 1.7. Furthermore, after a given mesh refinement, the box-spline interpolation seems to be the most accurate scheme.

In Figure 8, we can see that the performance converges quite quickly, for all the methods. It is also pretty obvious that, even if the splines are more accurate, the cost is higher than most of the Hermite Finite Element methods.

In Figure 9 (left), we represent, with respect to N_c, the number of points treated per μ-second, also called the efficiency, on the machine irma-hpc2, whose characteristics are: HP Proliant DL 585 G6, 4 processors AMD Opteron 6 Cores

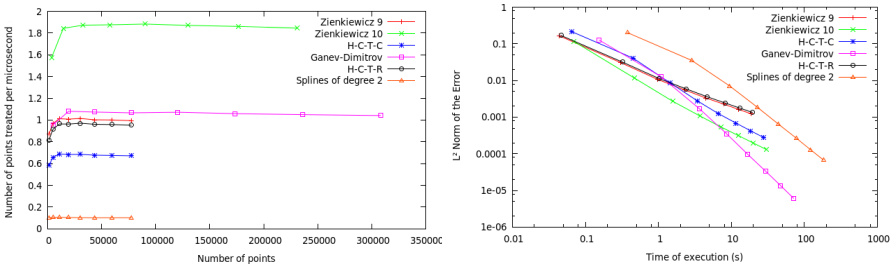

Figure 8: Comparison of performances for the circular advection test case (CFL = 2.)

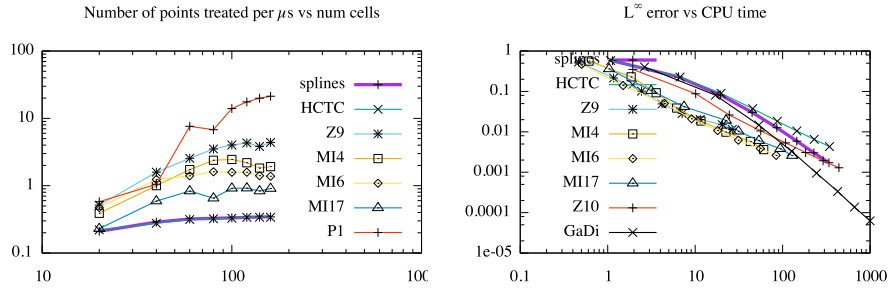

Figure 9: Comparison of performances for the circular advection test case (inclusion of Mitchell elements; CFL = 0.25; on machine irmahpc2)

at 2.8GHz, 128 Go RAM, 2.4 To of disk. All the algorithms are not at the same level of optimization: HCTC has not been optimized; P1, Z9 and splines have been optimized using different strategies. We think that splines could be further optimized. P1 which is the least costly method is also clearly the fastest method. We obtain around 1GFlops (when the efficiency is about 30) which is a correct value with respect to [27] for example. It is followed by the Z9 element, and the Mitchell elements which need also the computation of the mixed derivatives. Finally splines and HCTC have almost the same cost; note that previously, on Figure 8, splines were slower; HCTC were also faster, but the runs were not done on the same computer.

We then represent on Figure 9 (right), the error in L^∞ norm with respect to time. If GaDi finally wins, the Mitchell elements are among the best in the current implementation. Going higher in degrees leads to time overhead for the computation of the derivatives, this yields that MI17 is not the most competitive, whereas MI6, because it is less accurate, is better, as it is faster. Note as well that Z9 behaves really well. Box-splines are not as competitive, but the situation could change, by optimizing even further the code. We should notice that, for the Mitchell elements, the storage of the different derivatives is multiplied by 13, which leads to costly memory access. Whereas, for box-splines, this is not

the case (coefficients are computed when needed, making this method less costly memory-wise, but slower).

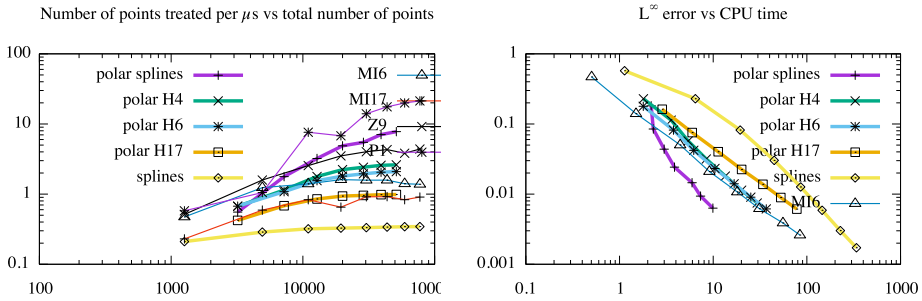

Figure 10: Comparison of performances for the circular advection test case and comparison with classical interpolation on polar mesh; CFL = 0.25; on machine irmahpc2

Firstly, in Figure 10 (left), we study the performance with classical interpolation schemes on polar mesh (we use cubic splines and Hermite of degree p; see e.g. [23]). The polar mesh is obtained by discretizing the annulus $\Omega = \{(r\cos(\theta), r\sin(\theta)),\ r \in [r_{\min}, r_{\max}] \times [0, 2\pi]\}$, with $r_{\min} = 10^{-5}$, $r_{\max} = 8$ using a grid of N_c cells in r and $2N_c$ cells in θ direction. In Figure 10 (left), we compare again the number of points advected per μ-second. We see that in this polar setting, the classical cubic splines method has a very good efficiency and the current Hermite interpolation H(p) has a decreased efficiency and this gets worse as the degree p increases. On the hexagonal grid, only the P1 interpolation has a better efficiency and Z9 can approach similar efficiency to the splines in polar geometry, while being better than the polar Hermite methods. On the other hand, the efficiency is quite comparable for the Hermite and Mitchell methods for a given degree p. Box-splines are still the less efficient for the moment.

Secondly, we compare the error in L^∞ norm with respect to the CPU time, see Figure 10 (right). Cubic splines in polar geometry outperforms the other methods. Choosing another test-case more favorable to the waste of points due to the polar geometry could change the situation. On the other hand, the second better method with respect to this diagnostic is MI6 , which is encouraging (it is even better than polar splines for very low resolution and it has a more stable and foreseeable behavior). We should again warn the reader that all these studies have been done for a given implementation and that the situation could change if all the methods are fully optimized.

We then look Figure 11 where the L^∞ error is plotted with respect to the number points, making again the comparison between polar and hexagonal mesh. For low number of points, box-splines are the least accurate but the situation changes when the number of grid points increases. Note that box-splines interpolation is not the complete analog of cubic splines interpolation on a polar grid, as it is quasi-interpolant. We can distinguish that MI17 is the most accurate, before box-splines become better. On the other hand, the accuracy

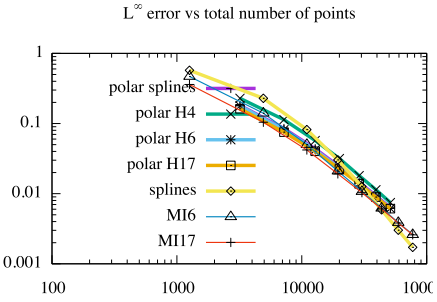

Figure 11: Order of convergence (inclusion of Mitchell elements) and comparison with classical interpolation on polar mesh; CFL = 0.25

remains quite similar between polar and hexagonal geometry, especially for the Hermite/Mitchell methods. Note also that the cubic splines method is almost the same as H6 (this is often the case, see [34]).

4.2 Guiding-center model - Diocotron instability test case

We consider here a guiding-center approximation of the 2D Vlasov-Poisson system. This also corresponds to the reduced gyrokinetic model obtained[19] when all quantities are homogeneous in the direction parallel to the magnetic field. Here the magnetic field is set to $B = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$. Then the model reads

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + E_\perp \cdot \nabla_{\mathbf{x}} \rho(\mathbf{x}, t) = 0 & \text{(28a)} \\ -\Delta \phi = \nabla \cdot E = \rho(\mathbf{x}, t) & \text{(28b)} \end{cases}$$

with $E = (E_x, E_y) = -\nabla \phi$ and $E_\perp = (-E_y, E_x)$.

By neglecting the effect of boundary conditions (here, we took null Dirichlet), the guiding-center model verifies the following properties:

1. Positivity of density ρ
$$0 \leq \rho(x, y, t).$$
2. Mass conservation
$$\frac{d}{dt}\left(\int_D \rho \, dxdy\right) = 0.$$
3. L^p norm conservation, for $1 \leq p \leq \infty$
$$\frac{d}{dt}||\rho||_{L^p(D)} = 0.$$
4. Energy conservation
$$\frac{d}{dt}\left(\int_D |\nabla \phi|^2 dxdy\right) = 0.$$

This model, is commonly used in 2D simulations to study the particle density, as it describes highly magnetized plasmas in the poloidal plane of a tokamak.

We chose here to study the diocotron instability [15]. The initial density is given by:

$$\rho_0(\mathbf{x}_\perp) = \begin{cases} (1+\varepsilon\cos(\ell\theta))\exp\left(-4(r-6.5)^2\right), & \text{if } r^- \leq \sqrt{x^2+y^2} \leq r^+, \text{ with } \theta = \text{atan2}(y,x). \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Here r^+ (respectively r^-) is the maximum (resp. minimum) radius of an annulus where ρ_0 is not null. θ is the radian angle given by (x, y). As for the parameters values, we take $\varepsilon = 0.001$, $r^- = 5$, $r^+ = 8$, $\ell = 6$, $dt = 0.1$ and the hexagonal step is $\frac{14}{160}$ with a radius of 14 and and a hexagonal parameter $N_c = 160$. In this part, we will not test the Z10 approach as it requires a special resolution of the Poisson equation that has not been implemented. Indeed, computing the values of the field at the center of the triangles can't be combined with the resolution at the vertices. Moreover to even the computational time of each method we chose to take $N_c = 80$ for the Ganev Dimitrov element as it results in the computations on a mesh with $N_c = 160$ (see Section 1.3.4).

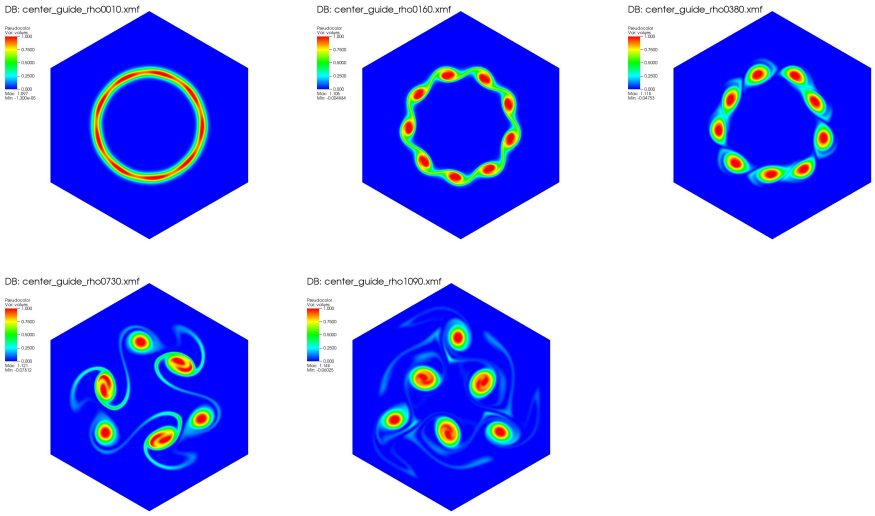

Figure 12: Time evolution of the guiding-center model with $\varepsilon = 0.1$, at times = 1, 16, 38, 73 and 109

After playing with the parameters, we note that 6 vortices is the main mode. In fact, if we take $\ell \neq 6$, with ε small enough, we still see the mode 6 appear. With ε big enough, i.e. at least 0.1, the modes different from 6 can be visible for a time but they are not stable, thus we see the fusion or the apparition of vortices until the sixth mode takes over. For instance, as illustrated by Figure 12, we

can see the ninth mode turning into the sixth mode by fusion of vortices. This instability can be explained with Figure 13. The influence of the geometry is clear as the potential is not round, but already deformed as a hexagon. This phenomenon is clearly caused by the boundary conditions (null Dirichlet) and the shape of the geometry; If other boundary conditions were imposed, we would be able to see results similar to what we can get in a polar mesh (where any given mode can be captured[15]).

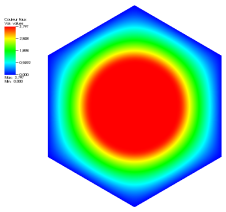

Figure 13: Potential at time zero for the guiding-center simulation

Remark 4.1 *When running the guiding-center model with test-case* (1)*, see Figure 12, no obvious differences are visible between the different interpolation methods, which makes the diagnostics all the more important to compare the results computed.*

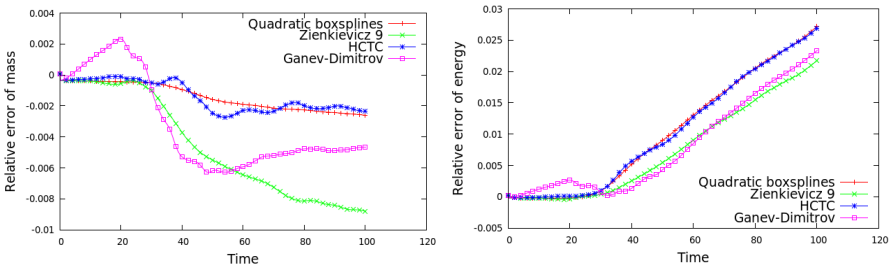

Figure 14: Time evolution of the relative error of mass and energy

After comparison of the diagnostics in Figures 14, 15, and 16, we see that the various interpolation methods give close results overall. They are similar in terms of positivity conservation, specially when comparing the Z9 elements to the splines; the other Hermite elements have globally a worse positivity conservation. We notice that if box-splines conserve better the mass, the Z9 approach conserves better the L^1 norm. Also we note that box-splines and the HCTC element give very near results whichever the diagnostic considered.

In Figures 17, 18, and 19, we consider the same diagnostics as before, but we include a comparison with the Mitchell elements. Note that the Ganev

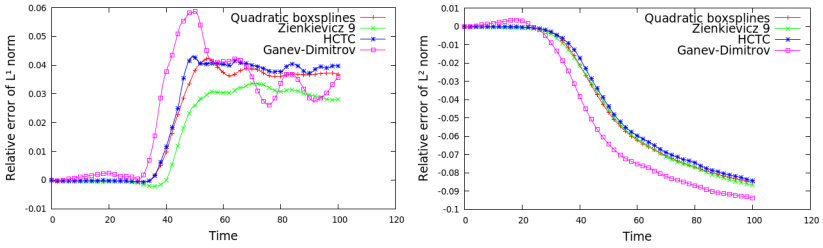

Figure 15: Relative error of L^1 and L^2 norms

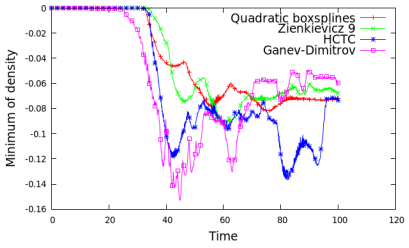

Figure 16: Time evolution of the density's minimum

Dimitrov has not the same degrees of freedom as the other methods. We define $N_{\text{tot}} = 2N_c$ for this method and $N = N_c$ for all the other methods. On Figure 17, we consider the case where $N_c = 128$, $\Delta t = 2^{-3}$. We then refine the mesh by 2, on Figure 18 and the time step by 2 on Figure 19, in order to see some effects of changing time and space resolutions.

We notice that the Ganev-Dimitrov FE interpolation has good accuracy when we use the same number of cells (e.g. $N = 128$ and $N_{\text{tot}} = 256$); but this is not fair as the complexity is not the same. So, we should compare, as done previously, using for example $N_{\text{tot}} = N = 128$ or $N_{\text{tot}} = N = 256$. In that case, we see that the method is no more competitive as the quantities are badly conserved in comparison to the other methods. The situation may be different if the grid is finer, but this then becomes more difficult to solve, as the Poisson solver would take much longer or take too much memory space, at least in the current implementation.

Concerning the mass, the box-splines method outperforms the other methods (probably due to their property of partition of unity). HCTC has near same accuracy, but it is more oscillating. Z9 is clearly less accurate. We notice that the Mitchell elements are better than Z9 and that increasing p leads to better results. With MI17, we are not at the same level of accuracy as the one for the box-splines, but we approach it. Note that the results are different on Figure 18, probably because the time step is too big with respect to the resolution in space. This can affect the convergence in time: the solution is more complex and a bigger time step leads to worse mass conservation. Furthermore, this may also explain the fact that MI4 has better mass conservation than MI17. For

energy conservation, results are quite similar, mainly, only Z9 behaves better. Energy (as mass) is better conserved, when the time step gets smaller. L^1 norm conservation is improved with the Mitchell elements, in particular with MI17 which is better than Z9; MI6 is better than HCTC and box-splines, but not than Z9.

We see that L^2 norm is better conserved when increasing the degree p of the Mitchell elements. HCTC, Z9 and MI4 are almost at the same level: slightly better than box-splines. Note that MI17 is almost at the level of GaDi which requires 4 times more point (see Figure 17). As previously mentioned, this is probably caused by the reconstruction of the normal derivatives (see Remark 1.7). Concerning the L^∞ norm, the best results are obtained for Mitchell's elements, in particular MI17. Box-splines and HCTC do not behave as good as the latter, and Z9 behaves also well. Finally, for the minimum density, HCTC is one of the worst method. Box-splines seem to behave the best. H17 or Z9 come close next, when the time step is small enough, otherwise huge negative values appear for all the methods, except box-splines.

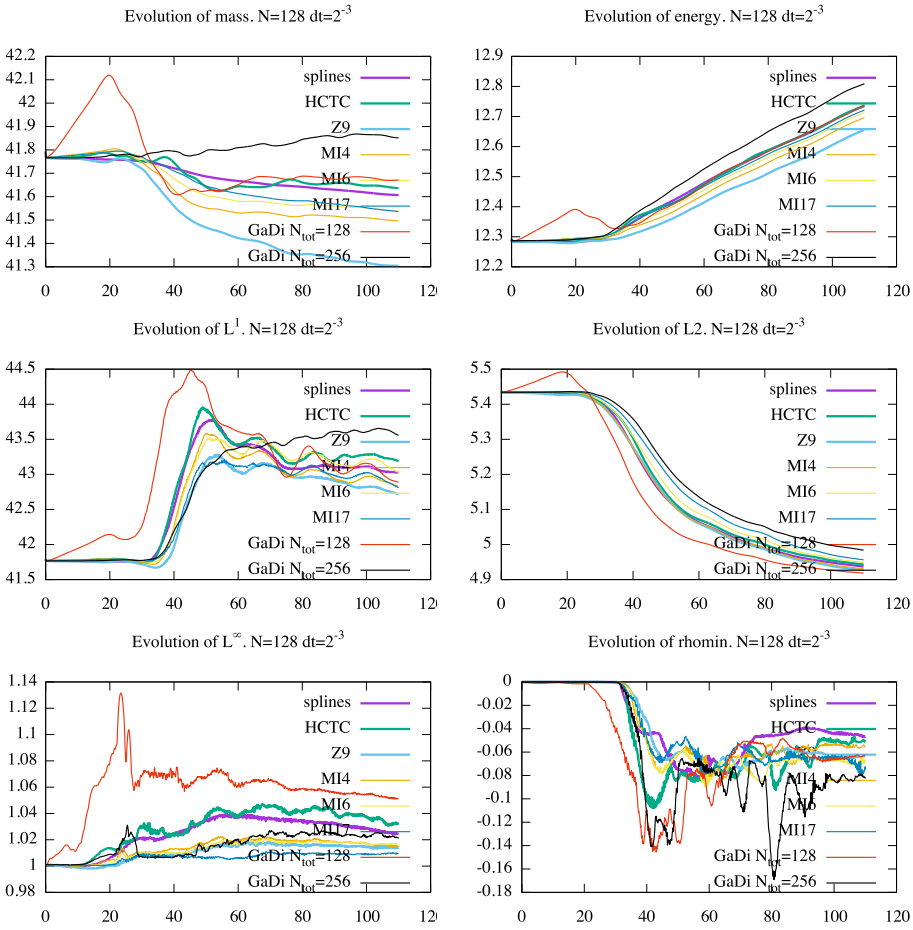

Figure 17: Guiding-center case. Evolution of mass, energy, L^1, L^2, L^∞ and minimum density

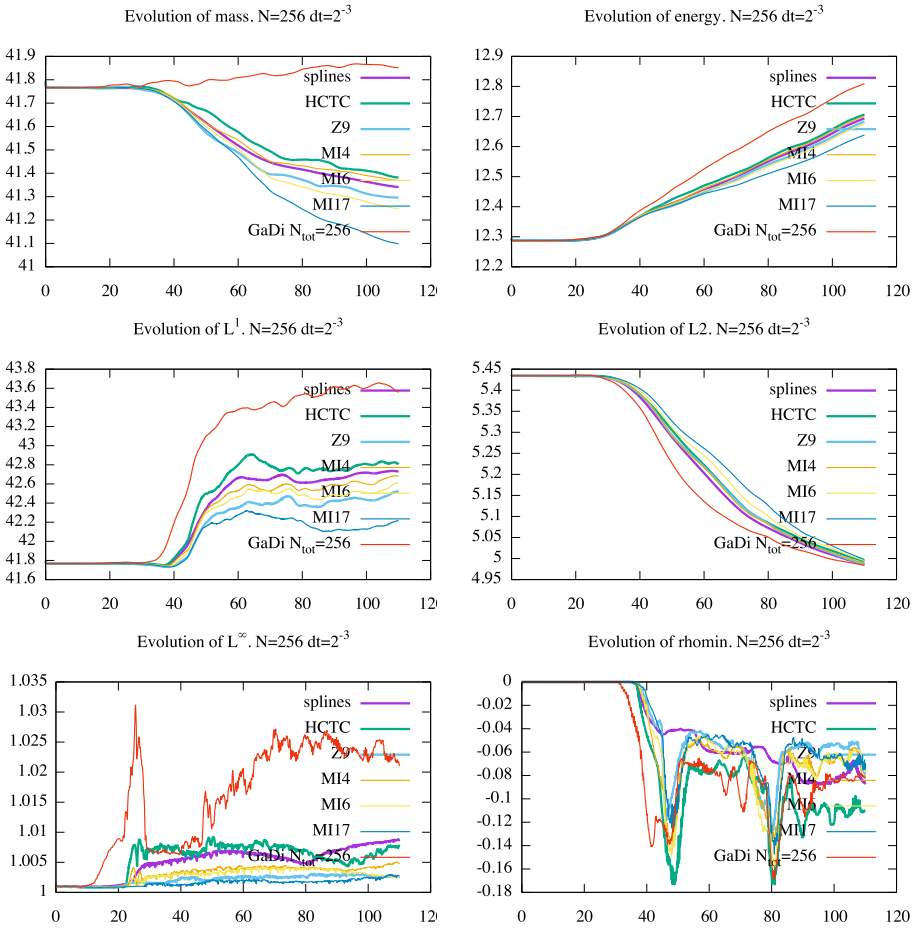

Figure 18: Guiding-center case. Evolution of mass, energy, L^1, L^2, L^∞ and minimum density

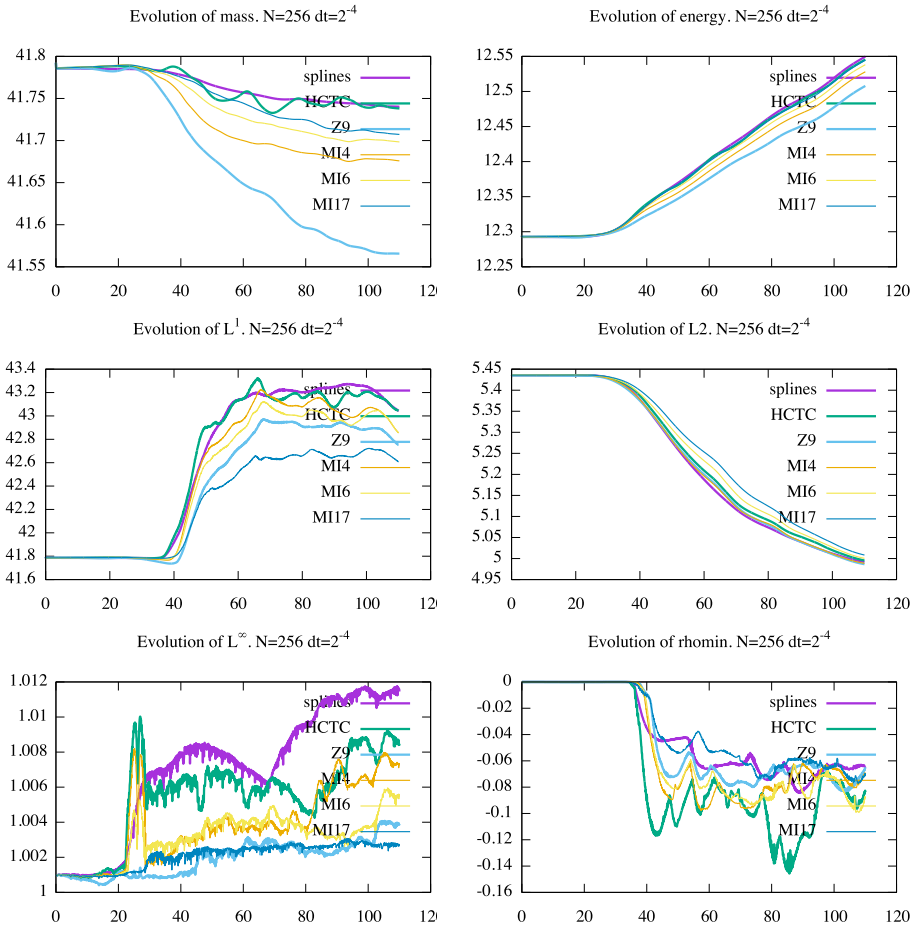

Figure 19: Guiding-center case. Evolution of mass, energy, L^1, L^2, L^∞ and minimum density

5 Conclusion

In this paper we tested Semi-Lagrangian schemes adapted to a hexagonal mesh. The strategies differentiated in the type of interpolation used: on the one hand, we developed an interpolation method based on box-splines –spline basis specific to the hexagonal mesh– and on the other hand, we introduced an interpolation method based on Hermite Finite Elements. Furthermore, we presented a Poisson solver based on finite differences on this mesh. The first simulations were made on the circular advection test case. This allowed to compare the order of the methods, as well as their efficiency. The splines approach seemed to be more accurate for finer grids, but at higher cost. Whereas, the Mitchell element, was the best method when studying the efficiency. Next, we simulated the 2D guiding-center model. The two methods yield comparable results. The interpolation using Hermite elements is always faster than the box-splines quasi-interpolation. Box-splines seem to have a better mass and positivity conservation, whereas the Mitchell elements conserved the norms far more accurately and the Z9 seem unbeatable regarding the conservation of energy. Nevertheless, none of the methods stood out from the others. Further research is needed to compare the two methods in more complex problems. Although overall, we believe the splines interpolation method could be optimized and therefore be undoubtedly favored for the circular advection and the guiding-center model when working with fine meshes.

References

[1] J. Abiteboul, G. Latu, V. Grandgirard, A. Ratnani, E. Sonnendrücker, and A. Strugarek. Solving the vlasov equation in complex geometries. *ESAIM: Proceedings*, 32:103–117, 2011.

[2] P. Angelino, X. Garbet, L. Villard, A. Bottino, S. Jolliet, P. Ghendrih, V. Grandgirard, B. McMillan, Y. Sarazin, G. Dif-Pradalier, et al. Role of plasma elongation on turbulent transport in magnetically confined plasmas. *Physical review letters*, 102(19):195002, 2009.

[3] J. W. Banks and J. A. F. Hittinger. A new class of nonlinear finite-volume methods for vlasov simulation. *Plasma Science, IEEE Transactions on*, 38(9):2198–2207, 2010.

[4] M. Bernadou. *Méthodes d'éléments finis pour les problèmes de coques minces*, volume 33. Masson, 1994.

[5] N. Besse. *Etude mathématique et numérique de l'équation de Vlasov non linéaire sur des maillages non structurés de l'espace des phases*. PhD thesis, Université Louis Pasteur, 2003.

[6] N. Besse and E. Sonnendrücker. Semi-lagrangian schemes for the vlasov equation on an unstructured mesh of phase space. *Journal of Computational Physics*, 191(2):341 – 376, 2003.

[7] C. K. Birdsall and A. B. Langdon. *Plasma physics via computer simulation*. CRC Press, 2004.

[8] E. S. Carlson, H. Sun, D. H. Smith, and J. Zhang. Third order accuracy of the 4-point hexagonal net grid. finite difference scheme for solving the 2d helmholtz equation. Technical report, 2003.

[9] P. Chatelain and A. Leonard. Isotropic compact interpolation schemes for particle methods. *Journal of Computational Physics*, 227(6):3244–3259, 2008.

[10] C. Cheng and G. Knorr. The integration of the vlasov equation in configuration space. *Journal of Computational Physics*, 22(3):330 – 351, 1976.

[11] L. Condat and D. Van De Ville. Three-directional box-splines: characterization and efficient evaluation. *IEEE Signal Process. Lett.*, 13(7):417–420, 2006.

[12] L. Condat and D. Van De Ville. Quasi-interpolating spline models for hexagonally-sampled data. *IEEE, Transactions on Image Processing*, 16(5):1195–1206, May 2007.

[13] L. Condat and D. Van De Ville. New optimized spline functions for interpolation on the hexagonal lattice. In *ICIP*, pages 1256–1259. IEEE, 2008.

[14] L. Condat, D. Van De Ville, and M. Unser. Efficient reconstruction of hexagonally sampled data using three-directional box-splines. In *ICIP*, pages 697–700. IEEE, 2006.

[15] N. Crouseilles, P. Glanc, S. Hirstoaga, E. Madaule, M. Mehrenberger, and J. Pétri. A new fully two-dimensional conservative semi-lagrangian method: applications on polar grids, from diocotron instability to itg turbulence. *The European Physical Journal D*, 68:1–10, 2014.

[16] C. de Boor, K. Höllig, and S. Riemenschneider. *Box Splines*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[17] F. Filbet and C. Prouveur. High order time discretization for backward semi-lagrangian methods. `https://hal.archives-ouvertes.fr/hal-01133854`, 2015.

[18] F. Filbet and E. Sonnendrücker. Comparison of eulerian vlasov solvers. *Computer Physics Communications*, 150(3):247–266, 2003.

[19] F. Filbet and C. Yang. Mixed semi-lagrangian/finite difference methods for plasma simulations, Sept. 2014. https://hal.inria.fr/hal-01068223.

[20] F. Golse and L. Saint-Raymond. L'approximation centre-guide pour l'équation de vlasov-poisson 2d. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 327(10):865 – 870, 1998.

[21] V. Grandgirard, M. Brunetti, P. Bertrand, N. Besse, X. Garbet, P. Ghendrih, G. Manfredi, Y. Sarazin, O. Sauter, E. Sonnendrücker, et al. A drift-kinetic semi-lagrangian 4d code for ion turbulence simulation. *Journal of Computational Physics*, 217(2):395–423, 2006.

[22] G. Guscaglia and V. Rua. Finite element methods for the stokes system based on a zienkiewicz type n-simplex. *Computer Methods in Applied Mechanics and Engineering*, 272:83–99, 2014.

[23] A. Hamiaz, M. Mehrenberger, H. Sellama, and E. Sonnendrücker. The semi-lagrangian method on curvilinear grids. `https://hal.archives-ouvertes.fr/hal-01213366`, October 2015.

[24] E. Kalnay. *Atmospheric Modeling, Data Assimilation and Predictability.* Cambridge university press, 2003.

[25] J.-M. Kwon, D. Yi, X. Piao, and P. Kim. Development of semi-lagrangian gyrokinetic code for full-f turbulence simulation in general tokamak geometry. *Journal of Computational Physics*, 283:518–540, 2015.

[26] T. Lyche, C. Manni, and P. Sablonnière. Quasi-interpolation projectors for box splines. *Journal of Computational and Applied Mathematics*, 221(2):416–429, 2008.

[27] M. Mehrenberger, C. Steiner., L. Marradi, M. Mehrenberger, N. Crouseilles, E. Sonnendrücker, and B. Afeyan. Vlasov on gpu (vog project). *ESAIM: PROCEEDINGS*, 43:37–58, 2013.

[28] R. M. Mersereau. The processing of hexagonally sampled two-dimensional signals. *Proceedings of the IEEE*, 67(6):930–949, 1979.

[29] A. Mitchell. An introductions to the mathematics of the finite element method. In *The Mathematics of finite elements and applications*, pages 37–58. J. R. Whiteman, 1973.

[30] T. Nguyen, K. Karčiauskas, and J. Peters. A comparative study of several classical, discrete differential and isogeometric methods for solving poisson's equation on the disk. *Axioms*, 3(2):280–299, 2014.

[31] A. Ratnani. Isogeometric analysis in plasmas physics and electromagnetism. In *Workshop on Higher Order Finite Element and Isogeometric Methods Program and Book of Abstracts*, page 64, 2011.

[32] R. Sadourny, A. Arakawa, and Y. Mintz. Integration of the nondivergent barotropic vorticity equation with an icosahedral-hexagonal grid for the sphere. *Monthly Weather Review*, 96(6):351–356, 2014/11/21 1968.

[33] 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*, 149(2):201–220, 1999.

[34] C. Steiner. *Numerical computation of the gyroaverage operator and coupling with the Vlasov gyrokinetic equations*. PhD thesis, IRMA, December 2014.

[35] R. Ulichney. *Digital halftoning*. MIT press, 1987.

[36] D. Van De Ville, T. Blu, M. Unser, W. Philips, I. Lemahieu, and R. Van de Walle. Hex-splines: a novel spline family for hexagonal lattices. *Image Processing, IEEE Transactions on*, 13(6):758–772, 2004.

[37] S. Zaki, L. Gardner, and T. Boyd. A finite element code for the simulation of one-dimensional vlasov plasmas. i. theory. *Journal of Computational Physics*, 79(1):184–199, 1988.