A new approach discretising the 2D poloidal plane of fusion devices

Laura S. Mendoza

Max-Planck-Institut
für Plasmaphysik

TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Mathematik
Lehrstuhl für Numerische Methoden der Plasmaphysik

A new approach discretising the 2D poloidal plane of fusion devices

Laura S. Mendoza

Vollständiger Abdruck der von der Fakultät für Mathematik der Technische Universität München zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigten Dissertation.

Vorsitzende:	Prof. Dr. Caroline Lasser
Prüfer der Dissertation:	1. Prof. Dr. Eric Sonnendrücker
	2. Prof. Dr. Francis Filbet
	3. Prof. Dr. Philippe Helluy

Die Dissertation wurde am 28. April 2017
bei der Technischen Universität München eingereicht
und durch die Fakultät für Mathematik
am 21. Juni 2017 angenommen.

Abstract

The GYSELA code is a non-linear 5D global gyrokinetic code which performs flux-driven simulations to solve the gyrokinetic Vlasov equation coupled with the Poisson equation. Its 3D spatial representation is limited to circular toroidal geometry (r, θ, φ). Currently the poloidal plane, a circular cross-section, is discretized with a polar mesh. Due to the singularity of this mapping on its origin, the geometry is discontinuous (with a hole in the center). Furthermore, the code is currently not adapted to simulations on D-shaped tori.

In this work, our aim is to test different solutions to generalize GYSELA's geometry definition. The solutions presented are all in a general curvilinear case, so that any geometry, however complex, can be simulated by mapping one or multiple patches to the final wished geometry. We decided to study two different approaches to solve this problem: on the one hand, using an IgA (Isogeometric Analysis) based on Non-Uniform Rational B-Splines (NURBS), which provide an exact representation of complex shapes, allowing us to "fill the hole" using a 5-patch mapping; on the other hand, using a Finite Element Method on a regular equilateral triangle mesh of hexagonal form, which can therefore also be seen as a mesh formed of nested hexagons, based on Box-Splines. We call this mesh, the hexagonal mesh, and is easily mapped to a circle (or a D-shaped plane) by a stretching without any singular points.

The Semi-Lagrangian scheme, used in GYSELA, consists of two steps: computing the characteristics feet and interpolating on those points. Both steps were adapted to the Multi-patch approach, as well as different techniques to treat the boundary conditions in between patches. Even when using an accurate approximation of the boundary condition for the interpolation, we could not prevent the appearance of numerical noise around the singular points. Therefore, the lack of singularities in the hexagonal mesh is more appealing. A Poisson FEM solver using box-splines was needed in order to solve the Vlasov-Poisson equation in the hexagonal domain, as well as an implementation of Nitsche's method to treat the boundary conditions. Finally, we also presented the different mappings of the hexagonal mesh to realistic geometries (circle and D-shape) and showed the results of an advection equation on the circular domain.

The GYSELA code is one of many examples of why we need Semi-Lagrangian codes adapted to complex geometries. Other examples from plasma physics (and further goals) are the X-point, the scrape-off layer or edge plasma, 3D representation of a Tokamak and Stellarator, etc. The results presented in this work, lead us to believe that the ideal solution could probably be a combination of the two concepts presented: keeping an Isogeometric Multi-patch polar mesh for the external crown of the poloidal cut, coupled with a mapped hexagonal mesh for the center of the domain.

Zusammenfassung

Gysela ist ein 5D globaler gyrokinetischer nichtlinearer Code. Es löst das gyrokinetische Vlasov-Poisson-System mit hilfe des Semi-Lagrange-Verfahrens. Sein drei dimensionale Raum wird durch eine kreisförmige toroidale Geometrie (r, θ, φ) beschrieben. Derzeit wird die poloidale Ebene mit einem Polargitter diskretisiert. Diese Koordinatentransformation ist im Nullpunkt singulär. So ist die Geometrie diskontinuierlich (mit einem Loch in der Mitte).

In der vorliegenden Arbeit, ist es unser Ziel, verschiedene Lösungen zu testen, um die Geometriedefinition von Gysela zu verallgemeinern. Alle Lösungen sind im allgemeinen krummlinige Koordinaten. So wird jede komplexe Geometrie durch die Abbildung eines oder mehreren Patches (Teilgebiete) definiert. Wir studieren zwei verschiedene Ansätze. Erstens, eine auf NURBS (Non-Uniform Rational B-Splines) und IgA (Isogeometric analysis) basiert: wir verwenden eine 5-Patch-Domain-Zerlegung für den Poloidalschnitt. Die zweite Methode verwendet als Gitter regelmäßige Sechsecke mit gleicher Kantenlänge, die jeweils in sechs Dreiecken unterteilt sind (hex-mesh) und Box-Splines (eine multivariate Verallgemeinerung von B-Splines).

Das Semi-Lagrange-Verfahren, das Gysela verwendet, besteht aus zwei Schritten. Zuerst, werden die charakteristischen Kurven der Lösung in jedem Gitterpunkt numerisch approximiert. Dann wird an jedem erhaltenen Punkt eine Interpolation durchgeführt. Diese beiden Werkzeuge wurden an unsere Ansätze angepasst. Für den Multi-patch-Ansatz, entwicklen wir auch verschiedene Techniken, um die Randwertaufgabe zu behandeln. Allerdings zeigten unsere Simulationen immer numerische Schwingungen um die singulären Punkte. Das hex-mesh hat keine singulären Punkte. So entwickelten wir eine Quasi-Interpolationsmethode auf diesem Gitter. Außerdem, wurde ein Poisson FEM-Löser mit Box-Splines benötigt, um die Vlasov-Poisson-Gleichung zu lösen. Wir haben auch die Nitsche-Methode angepasst, um die Randwertaufgabe zu behandeln. Schließlich präsentierten wir die verschiedenen Koordinatentransformationen des Hex-meshs zu realistischen Geometrien (Kreis und D-shape) und zeigen die Ergebnisse einer Advektionsgleichung auf einem Kreis.

Der Gysela code ist eines von vielen Beispielen, warum wir Semi-Lagrangian-Codes brauchen, die an komplexe Geometrien angepasst sind. Weitere Beispiele aus der Plasmaphysik und weitere Ziele sind unter anderem der X-Punkt, und die Randschichten (Scrape-off Layer). Die Ergebnisse in dieser Arbeit zeigen, dass die ideale Lösung vermutlich eine Kombination der beiden dargestellten Konzepte ist: ein isogeometrisches Multi-patch-Polargitter für die äußere Krone des Poloidschnittes, ge-paart mit einem hex-mesh für die Mitte des Gebietes.

Acknowledgments

During these last years, I met many incredible people that enriched this work, as well as my life. I wish to thank you all from the bottom of my heart.

Foremost, I would like to express my infinite gratitude to my advisor, Eric Sonnendrücker, who gave me this incredible opportunity. I deeply admire your insightfulness and vast knowledge. I will be forever indebted for how you trusted me all along. I'm grateful to Virginie Grandgirard, who also supervised my thesis. The weeks I spent in Cadarache were probably some of the richest moments during my PhD; I enjoyed greatly working next to her: her encouragement, pragmatism, and the numerous conversations about science and life, were essential to me and my thesis. Thanks to Ahmed, who introduced me to the IgA world.

Next, I would like to thank Francis Filbet and Philippe Helluy for the interest they showed on my research, for participating in the jury of this thesis, and for being referees of this manuscript. A special thanks to Philippe who gave me the opportunity to work with him even when I still needed to finish my manuscript and motivated me until the end. I would also like to thank Caroline Lasser for presiding the jury of this thesis.

For the following part, I am going to try to mention everybody in chronological order (and in the most appropriate language).

I had the pleasure to join the NMPP group almost since its beginning thus, I got to meet many bright minds who helped me along my journey. Especially thanks to Katharina, for her discipline and patience; Michael Kraus, for the cooking book recommendations and for explaining to me some of the IPP politics; Jakob an exemplar Bavarian who listened to me complain every day; Natalia, for her strength; Yaman, for being so meticulous and down-to-earth (and fun after work hours!). I am also grateful I got to meet Tiago, Omar, Adila… And for the most recent members: thanks to Herbert, my barefooted, happy office mate, to Camilla, the worst supermarket coach, and to Mustafa, my soul brother, to whom all I can say is: I wish you had arrived sooner! Of course, this section would not be complete without acknowledging Anneliese's hard work and genuine kindness. I would also like to thank some TOK members: Xin, for teaching me about her culture and for her cheerfulness, and Martin, who organized many movie nights to perfection.

To Francis, Abigail and David, a huge thanks. I will always treasure the months we spent together, I am honored to call you —exceptional people— my friends! Since you left, I have missed you deeply but I feel overjoyed to have seen you achieve so many of your goals. Thanks for Berlin, Budapest, the operas, the barbecues and, of course, the evenings at the Waikiki bar!

I also remember warmly the concerts, music discussions, bierfests, cocktails nights, *und den Deutschunterricht mit Dr. Will.* You bring out the hip-hop side of me and I love that. Thank you (and to Francis for the invitation) for the short trip to Wales.

Pour ta partie Emmanuel, je me permets de passer au français. J'ai tellement de choses à te remercier : les pauses café, les repas à midi, les voyages (Italie, Slovénie, Berlin, Turquie...), mais surtout pour ton soutien dans les moments les plus difficiles... Merci aussi pour le côté professionnel : l'article, le postdoctorat, les conseils, les idées. J'espère qu'on pourra continuer à parler (et se prendre la tête) autour d'une bière pendant beaucoup d'années et même, à travailler sur de nouveaux projets !

Merci à Jérémie, Didier, Capucine et François, d'avoir été ma famille dans cette belle ville qui est Garching, pour votre côté engagé, intellectuel, et français ! J'ai beaucoup grandi grâce à vous.

Ma reconnaissance va aussi à l'équipe de l'IRFM au CEA qui a fait mes séjours à Aix-en-Provence extrêmement agréables. Notamment François et Thomas. J'admirerai toujours le courage de Thomas dans sa vie personnelle et professionnelle. Merci à Hugo B. pour les conversations sur le sens de la vie, les plantes, la musique. Merci à Guillaume Latu pour sa bonne humeur et les remarques très justes sur ma recherche. Merci aussi à Jorge, Hugo A., Timothée, Clotilde, Damien, Claudia...

J'ai eu le plaisir d'assister à, non un, ni deux, mais trois CEMRACS. Je ne pourrai malheureusement pas nommer tout le monde, mais surtout un grand merci à José et Élise (mes fidèles compagnons, qui m'ont introduit à l'addiction à Friends), Maxime, Mehdi, Matthieu, Thomas, Meriem, Guillaume, Romain, Olivier, Frédérique, Marie, et Lorenzo ! Merci d'avoir rendu ces 6 semaines (x3) de travail intense agréables et inoubliables. Un chaleureux merci à Bruno et Nordine.

Ma vie à Garching/Munich n'aurait pas été complète sans Jérémie, tellement de choses à mentionner : les concerts, les randonnées, les voyages (Égypte, Norvège, Turquie, Berlin, Guatemala...), les brunchs... Bref, je ne pourrai jamais tout écrire en quelques lignes, merci d'avoir toujours été un homme exemplaire et m'avoir accompagné dans ce que j'espère restera l'époque la plus sombre de ma vie.

Je souhaite aussi remercier les doctorants et post-doctorants de Strasbourg qui m'ont reçu bras ouverts : Ranine, Stéphane, Amaury, Nico (sensei), Guillaume K. (petite-pince) et Marie. Mais surtout merci à Guillaume D. (plus vieille pince du monde) et Romain, qui écoutent mes péripéties sans se plaindre et sans me juger.

J'inclus Hussam dans la partie Strasbourg. Je suis contente d'avoir trouvé un autre esprit torturé par le sens de la vie ! Merci pour les cours sur les espaces de Krylov et d'arabe à 3 h du matin et surtout... شكرا على حنانك

Y para terminar, quisiera agradecer a las personas más importantes de mi vida : mi familia. A mi papá gracias por ser tan pilas, fiable, por tu paciencia y cariño y por tener confianza en mi. A mi mamá, mi inspiración número uno, le agradezco su apoyo, su comprensión, su amor infinito y por las incontables veces que me dio los ánimos que me faltaban. Ana y Sese, mis dos personas favoritas en todo el universo, no tengo palabras suficientes para agradecerles el amor, los consejos, por hacerme reír y por hacerme sentir constantemente orgullosa de ustedes y de mi. Son ustedes dos los que le dan sentido a mi vida. Mis compinches hasta el fin, los adoro !

Guatemala, gracias por darme la *chispa* ;
France, merci de m'avoir appris ce que veut dire la *liberté* ;
Deutschland, danke für die *Weltoffenheit*

Contents

List of Figures

1 Introduction

Introduction to plasmas and controlled fusion

What is plasma?

When sufficient energy is supplied to a gas, electrons are freed from their atoms and molecular bonds are broken. The gas becomes ionized with an overall null charge. We call this state of matter: plasma. It is actually one of the four states of matter that are observable in everyday life, besides gases, liquids, and solids. Different methods are used to energize a gas and get to the plasma state by knocking off the electrons of their places. For example, by bringing a gas to extremely high temperatures it is possible to obtain the fusion of hydrogen atoms to produce energy. Other methods include imposing the gas to lasers, microwaves, a powerful electric current and any other method that will subject the gas to a strong magnetic field. In this work, we will address hot plasmas.

Figure 1.1: Some examples of plasmas: the sun, the aurora borealis in Norway, and a plasma ball

The sun and other stars are a perfect example of astrophysical plasmas. Actually, plasma is the most common state of the known matter in the universe; it forms 99% of the visible universe [SF05, GB05]. On earth, we can find plasma

in natural phenomena like the aurora borealis or lightnings, but we also see it in man-produced products such as neon lights, plasma balls, TV screens and fusion power plants.

Thermonuclear Fusion

As the world energy consumption rises so does the greenhouse gases' atmospheric concentration (mainly from burning fossil fuels, the first source of energy in the world). Therefore, it is necessary to find an alternative energy source. Fusion energy is clean – it produces no greenhouse gases and no long living radiation – and it has a virtually unlimited supply in the ocean.

Fusion is obtained when two (or more) lighter particles, for example two isotopes of hydrogen: deuterium and tritium, are combined together to form an overall heavier one. There is a mass loss during the process hence the process releases energy in the reaction. For a Deuterium-Tritium (D-T) reaction, a Helium atom is obtained, as well as a high-energy neutron (See Fig 1.2).

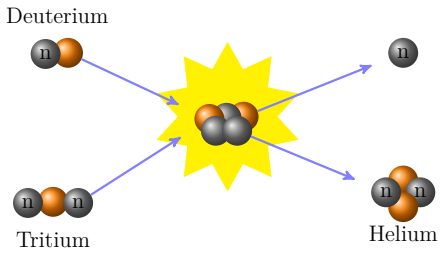


Figure 1.2: Scheme of the D-T fusion reaction

The main difficulty with fusion is to overcome the Coulomb forces. Indeed, two positive particles will repel each other, thus avoiding the nuclear strong force taking over and forming a new atom. To overpower the electric force, the fusing nuclei need to be high in energy. At really high temperatures, 1.2 billion Kelvin for the D-T reaction, the energy of the accidental collisions is high enough to obtain fusion. At this temperature matter is in the state of plasma.

Contrary to stars, where thermonuclear fusion is achieved by balancing the radiation with the gravitational forces, on earth, confinement of particles has to be imposed in another way. As we saw previously, plasma is an ionized gas thus responsive to magnetic fields. In the present work, we discuss this kind of confinement, as opposed to other methods like inertial confinement, electrostatic confinement, etc.

Fusion devices

We grasped the notion of plasma, fusion and the confining of plasma. Now, the big question is: what shape should the vessel containing the plasma have? Perhaps the simplest and most straightforward answer is the toroidal shape. When plasma is inside a magnetic field, it starts spiraling around it, because the force is perpendicular to both the velocity and the magnetic field. The toroidal shape will allow, in theory, for the field lines to go on the toroidal direction without touching the device's wall. It is also a simple shape to build and thus, the tokamak, a fusion reactor with toroidal shape, has become the favored geometrical shape for fusion reactors. It is important to notice that we need to twist field lines helically to compensate particle drifts, and thus the so-called concentric magnetic surfaces create, as we see in Figure 1.4.

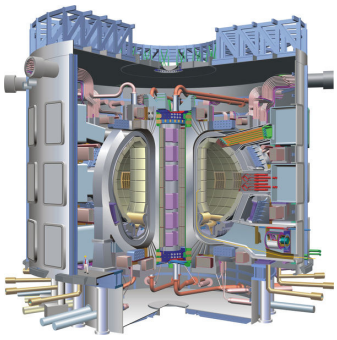


Figure 1.3: The ITER tokamak

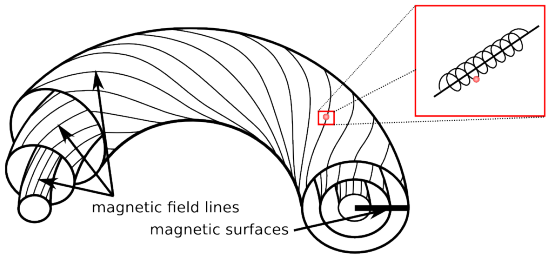


Figure 1.4: Schematic view of field lines in a tokamak

The ITER (International Thermonuclear Experimental Reactor) project is based on this geometry. ITER (See Fig 1.3) is being built in the south of France, in Cadarache, and is a joint project between the European Union, Japan, China, South Korea, Russia, the United States and India. The main idea of this project is to show that fusion reactors can actually be used for electricity production. At the moment, the closest a fusion reactor has been to produce more energy than what it was necessary for the experiment, is JET, a tokamak in England. The experiment had an amplification factor of $Q \sim 0.7$, whereas ITER is expected to obtain at least $Q = 10$.

The Vlasov equations

As physics experiments can be long and expensive, it is important to have a way of predicting (or simulating) future experiments. The first step to obtain an accurate computer simulation is to have the proper mathematical model that describes the physical system. To have a complete model of the plasma, we should know the

location and velocity of every single particle, as well as the electromagnetic field at every time given. Nonetheless, simpler models can be sufficient. Generally two big families of models are studied when studying plasma physics: Fluid and Kinetic models. Here, we will only address the kinetic models.

The Vlasov-Maxwell models

The Vlasov equation is often studied when modeling plasmas as it describes the evolution of the probability function, or distribution function, of charged particles at a given point, velocity and time in a collisionless plasma. The Vlasov equation is actually the statistical approach to describe the physical system of plasmas. It is usually coupled to the Maxwell equations to compute the self-consistent electric and magnetic fields, $\mathbf{E}$ and $\mathbf{B}$. Thus, the following system describes charged particles at any phase-space point.

$$\frac{\partial f_s}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f_s + \frac{q}{m}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \nabla_{\mathbf{v}} f_s = 0, \tag{1.1}$$

$$-\frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t} + \nabla \times \mathbf{B} = \mu_0 \mathbf{J}, \tag{1.2}$$

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \times \mathbf{E} = 0, \tag{1.3}$$

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}, \tag{1.4}$$

$$\nabla \cdot \mathbf{B} = 0. \tag{1.5}$$

Here f_s is the distribution function above mentioned for a given species, either ions or electrons. It depends on the position $\mathbf{x}$, velocity $\mathbf{v}$ of the particle on a given time t. We denote respectively by m and q the particles mass and charge. In Ampere's law (1.2), c is the speed of light. The constants μ_0 and ε_0 verify: $\mu_0 \varepsilon_0 c^2 = 0$. The total current density $\mathbf{J}$ and the charge density ρ are given by:

$$\rho(\mathbf{x}, t) = \int \sum_s q_s f_s(\mathbf{x}, \mathbf{v}, t)\ d\mathbf{v},$$

$$\mathbf{J}(\mathbf{x}, t) = \int \sum_s q_s \mathbf{v} f_s(\mathbf{x}, \mathbf{v}, t)\ d\mathbf{v}.$$

The distribution functions f_s depends on six variables, which makes the system, numerically, extremely expensive. Thus, we tend to favor the study of simpler models that still describe a plasma correctly, for example the Vlasov-Poisson system which is 6D but with a simplified Vlasov equation coupled to the Poisson equation.

The Vlasov-Poisson system

Another reason why the Vlasov-Maxwell is so complex is the dependency between the distribution function and the electromagnetic fields. The Vlasov-Poisson system is a reduced model of the Vlasov-Maxwell in the non-relativistic case ($\mathbf{v} \ll c$). It is a linearized model where the electric field $\mathbf{E}$ and the magnetic field $\mathbf{B}$ are independent of each other. Actually, the magnetic field is often simplified by $\mathbf{B} = (0,\ 0,\ B_0)^T$.

The equations (1.1) - (1.5) become

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f - \mathbf{E} \cdot \nabla_{\mathbf{v}} f = 0, \tag{1.6}$$

$$\Delta \phi = -\rho = -\int f(\mathbf{x}, \mathbf{v}, t), \tag{1.7}$$

$$\mathbf{E} = -\nabla \phi. \tag{1.8}$$

The Vlasov-Poisson system is often used to describe the Landau-Damping problem and many other electrostatic phenomena.

Conservation laws

All systems previously seen follow a certain number of conservation laws. It is important to know which laws the reduced model still conserves. This is also useful to verify codes and numerical simulations. The conservation properties verified by the Vlasov-Poisson system are the following:

1. Maximum principle (and positivity):

$$0 \leq f(\mathbf{x}, \mathbf{v}, t) \leq \max_{\mathbf{x},\mathbf{v}}(f(\mathbf{x}, \mathbf{v}, 0)). \tag{1.9}$$

   Follows from the method of characteristics (f preserved along characteristics)

2. Total momentum

$$\frac{d}{dt} \int \mathbf{v} f(\mathbf{x}, \mathbf{v}, t) \mathrm{d}\mathbf{x}\mathrm{d}\mathbf{v} = 0. \tag{1.10}$$

   Follows form multiplying by $\mathbf{v}$, and integrating phase-space

3. L^p norm conservation, for $p \in \mathbb{N}^*$ such that $1 \leq p \leq \infty$:

$$\frac{d}{dt} \left(\int_{\Omega} (f(\mathbf{x}, \mathbf{v}, t))^p \ \mathrm{d}\mathbf{x}\mathrm{d}\mathbf{v} \right) = 0. \tag{1.11}$$

   which includes the mass conservation.
   Follows from multiplying by f^{p-1} and integrating phase-space

4. Total number of particles (or mass conservation):

$$\frac{d}{dt}\left(\int_{\Omega} f(\mathbf{x},\mathbf{v},t)\,\mathrm{d}\mathbf{x}\mathrm{d}\mathbf{v}\right) = 0. \tag{1.12}$$

Follows from integrating in phase-space

5. Total energy:

$$\frac{d}{dt}\left(\frac{1}{2}\int \mathbf{v}^2 f \mathrm{d}\mathbf{x}\mathrm{d}\mathbf{v} + \frac{1}{2}\int (\nabla\phi)^2 \mathrm{d}\mathbf{x}\right) = 0. \tag{1.13}$$

Follows from using the Poisson equation, multiplying by $|\mathbf{v}|^2$ *and integrating phase-space*

In this work we will pay attention mainly to the positivity, L^p norms, and energy conservation. We will use these conservation laws to validate our methods and compare different schemes.

The Semi-Lagrangian method

When solving a Vlasov equation, one usually thinks of Lagrangian methods such as PIC [BL85]. However, these schemes are prone to numerical noise and converge slowly in $1/\sqrt{N}$ as the number of particles increases, typical of a Monte Carlo integration. Another option to solve the Vlasov equation is Eulerian methods like Finite Difference, Finite Element or Finite Volume methods [FS03, ZGB88, BH10]. The downside of this kind of method is that there is a numerical limit on the time step for explicit time stepping. With the attempt of overcoming the pitfalls of these methods, the Semi-Lagrangian method was introduced, first in numerical weather prediction (see [Kal03] and articles cited within it). Later, it was adapted to plasma simulations [SRBG99, CK76] and it is also used for gyrokinetics simulations of plasma turbulence [GBB+06, KYPK15, GAB+16].

This scheme consists on fixing an Eulerian grid in phase-space and following the trajectory of the equation characteristics in time to compute, by interpolation, 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 costly in high dimensions (6D phase-space mesh for a 3D simulation) where PIC methods still dominate.

Lastly, we can point out that there are different types of Semi-Lagrangian solvers (*e.g.* depending on the trajectories: backwards or forwards; depending on the degrees of freedom on which they are based: grid points, cell average, etc.). We have chosen here to use the classical Backward Semi-Lagrangian (BSL) method, since is the scheme used in the Gysela code.

The Backward Semi-Lagrangian method

Let us consider the following advection equation

$$\frac{\partial f}{\partial t} + \mathbf{a}(\mathbf{x}, t) \cdot \nabla f = 0. \tag{1.14}$$

We can compute its characteristics by solving the ODE

$$\frac{\mathrm{d}\mathbf{X}}{\mathrm{d}t} = \mathbf{a}(\mathbf{X}, t). \tag{1.15}$$

We call the unique solution $\mathbf{X}(t; \mathbf{x}, s)$ the characteristic of (1.14) at a time t which has as initial condition $\mathbf{X}(s) = \mathbf{x}$.

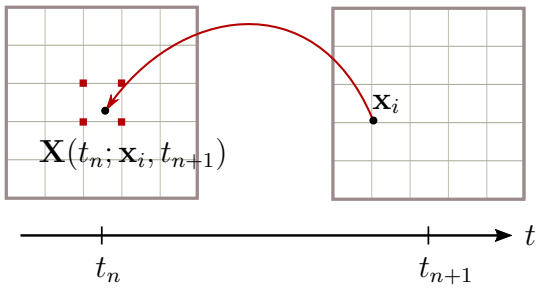


Figure 1.5: Backward Semi-Lagrangian scheme (example in 2D) – back tracing in time a mesh point $\mathbf{x}_i$ to its characteristic foot $\mathbf{X}(t_n; \mathbf{x}_i, t_{n+1})$

To compute f^{n+1}, the distribution function at the time t_{n+1}, the BSL scheme follows the characteristics back in time. For every mesh point $\mathbf{x}_i$, we compute the characteristic origin $\mathbf{X}(t_n; \mathbf{x}_i, t_{n+1})$. Knowing that the distribution function is conserved along the characteristics line, we can write

$$f^{n+1}(\mathbf{x}) = f^n(\mathbf{X}(t_n; \mathbf{x}_i, t_{n+1})). \tag{1.16}$$

At the time step t^n, we suppose that f is only known on the grid points. However, generally, $\mathbf{X}(t_n; \mathbf{x}_i, t_{n+1}) \notin \{\mathbf{x}_j,\ \forall j = 0 \dots N_{tot}\}$, where N_{tot} is the total number of mesh points. In other words, the origin of the characteristic is generally not a mesh point. Thus, the scheme requires an interpolation step. See Figure 1.5. The algorithm has to be accurate and efficient. Thus, the localization of the origin of the characteristics should be as fast as possible. For this reason, we choose to apply Semi-Lagrangian methods only on regular meshes.

Remark 1. *The computation of the origin of the characteristics and the interpolation step are the major sources of error in a low-dimensional Semi-Lagrangian scheme. Regarding the interpolation, we can cite the work in [BM08], where the authors compare different interpolation methods. The results show that the*

cubic-spline interpolation is advised for the best compromise between efficiency and accuracy. While treating higher dimensionality problems, typically 5 or 6D, the interpolation step becomes too expensive. Cheng and Knorr [CK76] introduced a solution consisting of a time-splitting approach. However, the splitting procedure becomes another major source of error.

Now that we presented the general context, model and the main idea of the scheme used to solve it, the next natural step is to present the code that represents the motivation of this manuscript.

The GYrokinetic SEmi-LAgrangian code

The non-linear 5D global gyrokinetic full-f code GYSELA [GAB+16, GBB+06] was created for flux-driven simulations of Ion Temperature Gradient (ITG) in tokamak plasmas. The code is being developed by the CEA of Cadarache, France, and other partners including the IPP, INRIA, and the University of Strasbourg. It is 5D meaning three-dimensional in space and two-dimensional in velocity. The time evolution of $\bar{f}$ is governed by the 5D collisional gyrokinetic equation described by Brizard and Hahm [BH07]. Here, we consider a simplified version of this model, where we neglected the right-hand side and considered the modified magnetic field simply as $\mathbf{B}^* = (0,\ 0,\ B_0)^T$, where B_0 corresponds to the magnetic field on the magnetic axis. The model becomes,

$$\frac{\partial \bar{f}}{\partial t} + \frac{\mathrm{d}r}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial r} + \frac{\mathrm{d}\theta}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial \theta} + \frac{\mathrm{d}\varphi}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial \varphi} + \frac{\mathrm{d}v_{||}}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial v_{||}} = 0 \tag{1.17}$$

where $\bar{f}(r, \theta, \varphi, v_{||}, \mu, t)$ is the distribution function averaged over the cyclotron motion and depending on the toroidal coordinates (r, θ, φ), the velocity parallel to the magnetic field $v_{||}$, the invariant magnetic momentum μ, and the time t. The gyrokinetic equation is self-consistently coupled to the quasi-neutrality equation

$$-\frac{1}{n_0(r)}\nabla_\perp \cdot \left[\frac{n_0(r)}{B_0\omega_c}\nabla_\perp \phi\right] + \frac{e}{T_e(r)}\left[\phi - \langle\phi\rangle\right] = \frac{1}{n_0(r)}\left[n_{Gi}(r,\theta,\varphi) - n_{Gi_{eq}}(r,\theta)\right] \tag{1.18}$$

with n_0 the particle density, B_0 the magnetic field computed at the magnetic axis, ω_c the ion cyclotron frequency, $-e$ and $T_e(r)$ are respectively the electron charge, and the electron temperature. ϕ is the electric potential and $\langle\phi\rangle$ its average on the magnetic flux surface. And lastly, we define the ion guiding-center density in Cartesian coordinates by $n_{Gi}(\mathbf{x}, t) = \int \mathbf{J}_v \, \mathrm{d}v_{Gi||} \, \mathrm{d}\mu \ \mathcal{J}_\mu \cdot \bar{f}(\mathbf{x}, \mathbf{v}, t)$, with $\mathcal{J}_\mu$ the gyro-average operator and where the Jacobian in the velocity space is given

by $\mathbf{J}_v = 2\pi B^*_{||}/m$. We define its correction term, $n_{Gi_{eq}}$, by taking $\bar{f}$ as an equilibrium distribution function $\bar{f}_{eq}$. For a detailed explanation on the terms and the model, we refer the reader to the work of V. Grandgirard, Y. Sarazin *et al.* [GSG+06, GS12, GAB+16].

GYSELA is based on some key physical assumptions (adiabatic electrons, electrostatic approximation, etc.) but the one that we will focus on is: the simplified magnetic geometry. The magnetic surfaces are considered to be concentric and circular (much like in Figure 1.4). The embedded closed magnetic flux surfaces play an important role and introduce an important anisotropy [AGV+09]. For this reason, one gets favorable numerical properties when grid points align on the concentric magnetic flux surfaces. Furthermore, GYSELA is based on the Semi-Lagrangian method. Thus, in a poloidal cut, the space grid is simply a field-aligned polar grid.

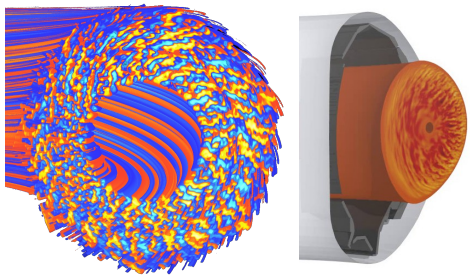


Figure 1.6: An example of a GYSELA simulation (left) where a section around the origin was not taken in consideration due to its singularity. On the right, a similar example of a GENE simulation

Despite the simplicity of the polar grid, the mesh presents a big disadvantage: its singularity at the center. This means that the Jacobian of the transformation that maps Cartesian into polar coordinates is null at the origin. In GYSELA, as some other (plasma) codes (*e.g.* GENE, GKW), a simple solution was found: cutting out an internal disk out of the domain. The discretized geometry becomes then discontinuous: an annulus in polar coordinates. Although the plasma density near the origin is lower than in the core of the annulus, the density is not entirely null. In fact, while back-tracing a particle, it might be that its origin falls in the non-represented area. When this happens, the particle is "lost" in the void.

Some codes inject new particles from the interior, to compensate for this loss (and maintain mass conservation). Nevertheless, this is far from optimal. The objective of this work is to find an alternate solution to this problem. Furthermore, we will take advantage of this upgrade of the geometry –passing from an annulus to circle– to present a solution more flexible to adapt to more complex geometries.

Outline of this manuscript

Different strategies have been implemented to avoid this singularity. We can cite among others: the iso-parametric analysis approach done by J. Abiteboul *et al.* [ALG+11] and A. Ratnani [Rat11] or N. Besse and E. Sonnendrücker's work with unstructured meshes [BS03]. The methods presented in these papers are particularly interesting. They avoid singularities and they are also adaptable to more complex geometries. However, even if these two approaches are different, they share the limitations due to the numerical complexity, the advection of the derivatives and, the localization of the origin of the characteristics.

Nonetheless, one aspect typically associated with iso-parametric analysis, the Multi-patch Approach, draws our attention. Actually, applying this concept means the conservation of a field-aligned mesh on the external part of the geometry and the introduction of a new grid for the core. Furthermore, using the main tools of IgA[1] and CAD[2], we should also win in the process the capability of adapting the discretization to more complex geometries. Thus, we will dedicate the following chapter to the basic tools for introducing this approach in our context. We will focus on the spline families used throughout in this thesis (B-Splines, NURBS and Box-splines), and introduce the key properties of these functions, as well as how we will use them in our work. The application of this technique, and some simulations using the Multi-patch Approach can be found in Chapter 3.

At some point, half along this thesis, we came across the uniform hexagonal mesh. The latter is composed of equilateral triangles and does not contain any singularities, unlike polar meshes. Thus, this mesh combined with a coordinate transformation is an alternative discretization of a poloidal plane. We explore this solution in Chapter 4.

Nevertheless, before getting into the details of any of the two approaches, we should start with the basis, presented in Chapter 2. The tools introduced therein should be useful for both Chapters 3 and 4.

[1] Isogeometric analysis, see next chapter.
[2] Computer Aided Design, see next chapter.

2

Computer Aided Design and Isogeometric Analysis

A common error encountered in numerical simulations when solving partial differential equations (PDE) is to neglect the design of the geometry. Computer Aided (Geometric) Design (or CAD for short, resp. CAGD), introduced in 1970, is a branch of study aiming to discretize a domain as accurately as possible without compromising computer resources such as computation time and memory space. It usually involves at least three main points: the construction of a specific mesh, solving any problems of continuity, and reducing the computational costs due to the geometrical representation. Parametric curves such as B-splines or NURBS are common tools in CAD. Isogeometric Analysis (IgA) [HCB05, CHB09] was born from the willing of incorporating CAD into Finite Element Analysis (FEA). Its key concept is to consider the same basis functions used for the FEA as the ones used for the geometry design.

There are many different approaches using this technique, they may vary on the basis functions used, or on how to treat the boundary conditions. Nevertheless, globally they all share the same global scheme for the mesh generation: the geometry is approximated by a coordinate transformation from an orthogonal rectangular grid to a more complex discretization matching the physical domain. Thus, firstly we will focus on this point and move latter to the different basis functions that we use throughout this manuscript.

Coordinate transformations and meshes

In this section, we will introduce the general principle of a coordinate transformation –the notations, the basic notions, and an example– as well as another type of

mesh that has not been used before for the CAD approach (at least in the plasma physics community).

Coordinate transformations

When a scheme is specific to a discretization of a precise geometry, it gives some advantages, mostly, in terms of optimization. Whereas, when a scheme is based on a Cartesian mesh coupled to a coordinate transformation, the method is globally independent of the domain that it is being applied to. Thus, it gains a flexibility that is imperative when developing a code implemented for different domains.

In Figure 2.1 we made a scheme of a classical coordinate transformation where a logical domain, defined by a patch $\mathcal{P}$, is mapped into a physical domain Ω. Following the most common conventions of the Isogeometric Analysis [CHB09] (IGA) a patch is a rectangular (or Cartesian) mesh that is mapped to a physical domain of curvilinear coordinates. In the patch, the coordinates (η, ξ) are equally spaced, such that: $\eta_{i+1} = \eta_i + \frac{1}{N_1} \quad \forall i = 0 \dots N_1 - 1$ and $\xi_{j+1} = \xi_j + \frac{1}{N_2} \quad \forall j = 0 \dots N_2 - 1$; N_1 and N_2 being respectively the number of points in η and ξ directions. The direct coordinate transformation $F : \mathcal{P} \to \Omega$ is given by $F(\eta, \xi) = (x, y)$. Finally, let F^{-1} be the inverse mapping.

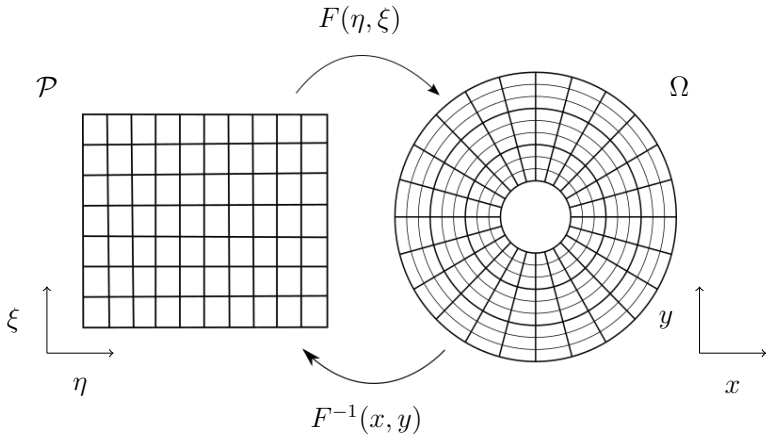


Figure 2.1: Representation of a polar coordinate transformation and its notations

The Cartesian to polar coordinates is represented above, which is by far the most common transformation in 2D. In 3D, we can mention the spherical or cylindrical transformations. In this manuscript, we use a polar mesh, but also other more sophisticated grids that we discuss in Chapter 3. Nevertheless, throughout all this chapter, as well as most of the studies being made in this domain, all transformations are based on the Cartesian grid. As we mentioned in the introduction, we were also interested in another regular mesh is mapped to a circular domain

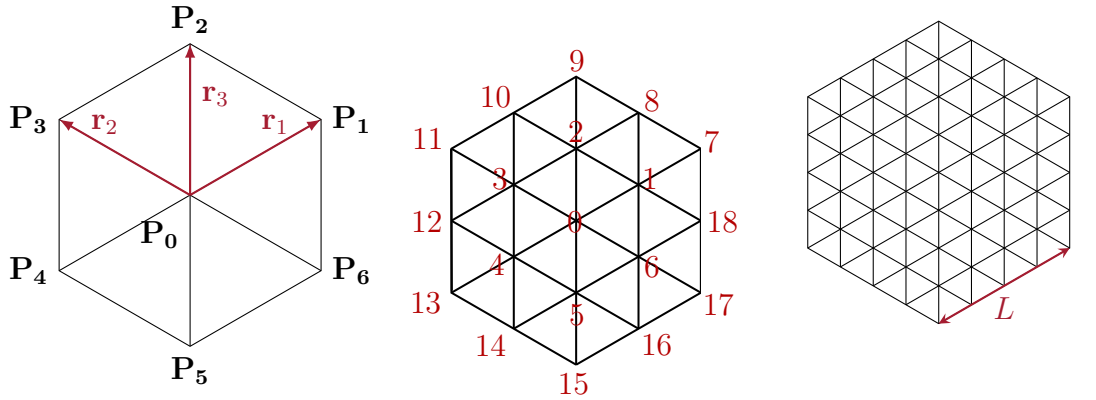
without any singularities. Namely, the hexagonal mesh, introduced in the next section.

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

$$\mathbf{r_1} = \begin{pmatrix} \sqrt{3}/2 \\ 1/2 \end{pmatrix}, \qquad \mathbf{r_2} = \begin{pmatrix} -\sqrt{3}/2 \\ 1/2 \end{pmatrix}, \qquad \mathbf{r_3} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.1}$$

The 2D lattice sites are obtained by the product $\mathbf{Rk}$ with the matrix $\mathbf{R} = (\mathbf{r_1}\ \mathbf{r_2})$ and the vector $\mathbf{k} = (k_1, k_2)^T \in \mathbb{Z}$. To obtain the mesh exactly as in Figure 2.2, we need to define a few extra parameters: an origin, denoted by $\mathbf{P}_0 = (x_0, y_0)$, a radius L (the distance between the origin and any external vertex) and the number of cells N_c on any radius.



(a) Hex-mesh with $N_c = 1$ **(b)** Hex-mesh with $N_c = 2$ **(c)** Hex-mesh with $N_c = 3$

Figure 2.2: Three hexagonal lattices with different mesh steps, the vectors $\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}$ that generate such meshes, and its point indexation

The mesh is based on uniform hexagons of the first type,[1] see [Uli87]. 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 2.2b). We will denote $\mathbf{P}_i$ the point of global index i, Cartesian coordinates $\mathbf{x}_i = (x_i, y_i)$ and

[1] The six main vertices of a first type hexagon are respectively at the angles ($\pi/6$, $\pi/2$, $5\pi/6$, $-5\pi/6$, $-\pi/2$, $-\pi/6$), see Figure 2.2. Whereas type two hexagons have the six main vertices at angles (0, $\pi/3$, $2\pi/3$, π, $-2\pi/3$, $-\pi/3$), equivalent to a regular hexagon of type one after a $\pi/6$ rotation

hexagonal coordinates $\mathbf{k}_i = \mathbf{R}^{-1}(\mathbf{x_i} - \mathbf{x_0})$. We notice that with these notations, $\mathbf{k}_0 = (0, 0)$, regardless of the values of $\mathbf{P}_0$, L or N_c. Likewise, $\mathbf{k}_1 = (1, 0)$, $\mathbf{k}_2 = (1, 1)$, and so on. For simplicity, we suppose $\mathbf{x_0} = (0,\ 0)$ throughout the rest of this thesis.

Besides the fact that the hexagonal mesh contains no singularities, its regularity allows us to locate the characteristics origins by taking three integer values, similarly to what is done on Cartesian grids for which only two integer values are needed. We will see the adaptation of the Semi-Lagrangian method to the hexagonal mesh in section 4. 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 [BM08]. Using the hexagonal lattice, B-splines do not exploit the isotropy of the mesh (for more information see [Mer79]) and are defined by a convolution in 2D, which can't be done for this mesh. Therefore, we need to introduce another basis function that could exploit this geometry.

After presenting the better-known basis functions that we used, B-splines and NURBS, we proceed to introduce the Box-splines, a more general type of splines that can be used for this hexagonal mesh.

Splines families: B-splines and NURBS

There exists a manifold of basis functions, hence at least as many approaches for both the domain decomposition and the solving of partial differential equations. In this work, we decided to consider one of the most recent families of basis functions, introduced in numerical analysis already in 1946 [Sch46], but that in recent years has grown in importance: the spline family. We will start with the well-known B-splines (see subsection 2), followed by the popular type of splines in the IgA community, NURBS (see subsection 2), and finally the most general form of splines, the Box-Splines (see subsection 2).

B-Splines

A basis spline, commonly known as B-spline, is a piecewise-polynomial curve [DB78]. B-splines are usually preferred for polynomial interpolations to Lagrange polynomials as they are not susceptible to Runge's phenomenon[2] and to Bezier curves considering that the entire Bezier interpolant has to be recomputed when moving a single control point.

[2]Runge's phenomenon is equivalent to the Gibbs phenomenon in the Fourier series approximation. It corresponds to the apparition of oscillations when interpolating with high order polynomials on an equidistant set of points [Run01].

Definition 1: B-spline of degree d centered in j

Let U be a sequence of $m+1$ knots in $[0,1]$, such that

$$0 \leq u_0 \leq u_1 \leq \cdots \leq u_m \leq 1. \tag{2.2}$$

The B-spline of degree d center in j is defined by recursion as

$$\begin{aligned} B_j^0(u) &= \begin{cases} 1 & \text{if} \quad u_j \leq u < u_{j+1} \\ 0 & \text{otherwise} \end{cases}, \\ \text{and} \quad B_j^{d+1}(u) &= \frac{u-u_j}{u_{j+d}-u_j} B_j^d(u) + \frac{u_{j+1}-u}{u_{j+d+1}-u_{j+1}} B_{j+1}^d(u). \end{aligned} \tag{2.3}$$

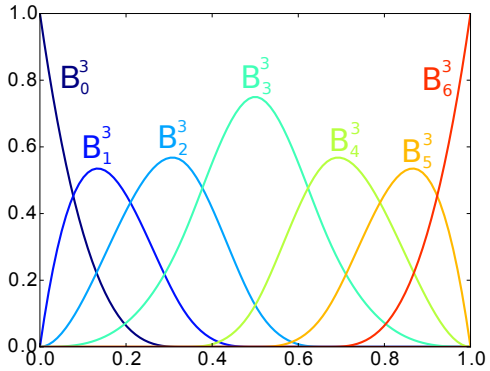


Figure 2.3: Cubic splines with knots $\mathbf{u} = \{0,\ 0,\ 1/3,\ 1/2,\ 2/3,\ 1,\ 1\}$

Another advantage of using B-splines is that they are well suited for the Semi-Lagrangian method.[3] This is due to the following properties.

- B-splines are positive: $B^d \geq 0, \quad \forall d \geq 0$;
- They are piecewise polynomials of degree d;
- The support of a B-spline is compact;
- B-splines naturally form a partition of unity: $\sum_j B_j^d(x), \quad \forall d$ and $\forall x$.

[3] As mentioned in Remark 1 we obtain the best compromise between cost and quality by using cubic B-spline for the interpolation step in the SL scheme.

Most of the properties can be visualized in Figure 2.3. As we saw in Section 1, positivity conservation (from the maximum principle) is important in our context. Thus, it is important to use basis functions that help to preserve this positivity. The second property leads to smooth functions which should ensure smooth interpolants, while a compact support makes for efficient codes. Finally, partition of unity play an important role when studying the conservation properties.

Furthermore, in recent years, B-splines have been introduced for the IgA approach by Manni *et al.* in 2011 [MPS11], but the most common basis functions in the IgA approach are NURBS, which we will introduce in the next section.

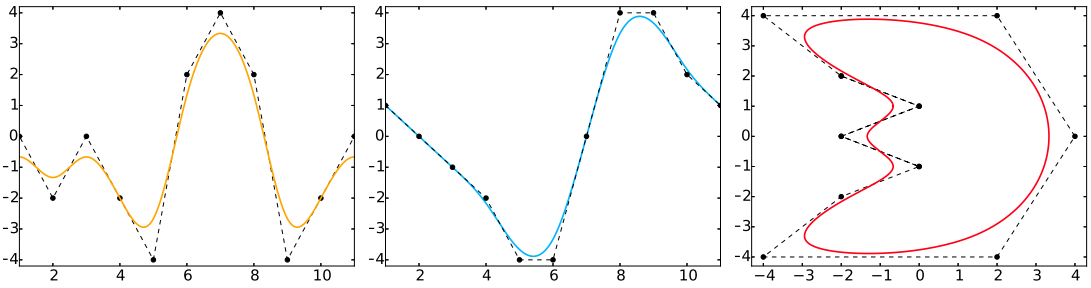


Figure 2.4: Three examples of B-spline curves (solid) and their control points (dashed)

From Definition 1 we can describe more complex objects. Starting with a B-spline curve (see Figure 2.4).

Definition 2: B-spline curve of degree d

A B-spline curve $\mathbf{C}\colon [0,1] \to \mathbb{R}^d$ of degree d is defined by

$$\mathbf{C}(u) = \sum_{j=0}^{m-d-1} B_j^d(u)\mathbf{P}_j\,, \tag{2.4}$$

where $\mathbf{P}_j$ are the $m-d$ control points and B_j^d the B-spline of degree d defined on the m knots $\{u_i\}$.

Going one dimension higher, we can define B-spline surfaces.

Definition 3: B-spline surface

A B-spline surface is defined by the tensor product of B-spline curves of degree

d_1 and d_2, such that

$$\mathbf{S}(u,v) = \sum_{j=0}^{m-d_2-1} \sum_{i=0}^{l-d_1-1} B_j^{d_2}(v) B_i^{d_1}(u) \mathbf{P}_{i,j}\,,$$

where $\mathbf{P}_{i,j}$ is a net of control points of size $((m-d_2)\times(l-d_1))$. $B_i^{d_1}$ and $B_j^{d_2}$ are defined respectively on the l (resp. m) knots $\{u_i\}$ (resp. $\{v_i\}$), following the definition in (2.2).

Another interesting point about B-splines is that their derivatives are easy to compute. This property is not directly used in this work, thus for more details we refer to [DB78, PT97]. Finally, let us introduce a more general type of splines than B-splines, NURBS. They are frequently used in IGA, and play an important role all along the first part of this thesis.

Non-Uniform Rational B-splines

Non-uniform rational basis splines (NURBS) are a mathematical tool used to define and represent curves and surfaces. The concept is the same used for generating B-spline curves and surfaces, associated with the concepts of tensor products (for surfaces) and rational Bezier curves. Using NURBS allows to have more precision and is easier to adapt to different shapes. For example, the representation of a circle is exact using NURBS whereas it is approximative with B-splines or Bezier curves. In fact, all conic sections can be exactly represented with NURBS. For the literature about NURBS we suggest [PT97].

Definition 4: NURBS curve of degree d

To define a NURBS curve, we need a set of control points $\{P_i\}$, their associated weights $\{w_i\}$, the d-th degree B-splines $\left\{B_i^d\right\}$ and a knot vector $U = \{u_0, u_1, \cdots, u_m\}$ such that

$$0 \leq u_0 \leq u_1 \leq \cdots \leq u_m \leq 1.$$

A NURBS curve is thus defined by

$$C(u) = \frac{\sum_{i=0}^{m-d-1} B_i^d(u) w_i P_i}{\sum_{i=0}^{m-d-1} B_i^d(u) w_i} \tag{2.5}$$

If we wish to extend the approach from 1D to 2D, the procedure is the same as with B-splines – using a tensor product between two 1D NURBS functions.

Definition 5: NURBS surface

Let U and V be two sequences of respectively $\ell+1$ and $m+1$ knots in $[0,1]$, such that

$$\begin{cases} 0 \leq u_0 \leq u_1 \leq \cdots \leq u_\ell \leq 1, \\ 0 \leq v_0 \leq v_1 \leq \cdots \leq v_m \leq 1. \end{cases} \tag{2.6}$$

A NURBS surface function can be written as

$$S(u,v) = \frac{\displaystyle\sum_{j=0}^{m-d_2-1} \sum_{i=0}^{\ell-d_1-1} B_j^{d_2}(v) B_i^{d_1}(u) w_i P_i}{\displaystyle\sum_{j=0}^{m-d_2-1} \sum_{i=0}^{\ell-d_1-1} B_j^{d_2}(v) B_i^{d_1}(u) w_i}, \tag{2.7}$$

where $B_i^{d_1}$ and $B_j^{d_2}$ are B-spline curves respectively of degree d_1 and d_2 associated with the knots vectors U and V.

Without going into details, we want to give in this section the formulas for the derivatives of a NURBS curve and surface. Indeed, curve and surface derivatives are essential for some algorithms. For example, the computation of the Jacobian matrix of a transformation calls for the derivative values of a NURBS curve. In our context, this will be particularly useful when computing the slopes for the boundary conditions in the Multi-patch Approach. However, the literature on this subject is already quite rich. We can cite [DB78, PT97, CHB09] for the details and proofs. Let us give the main points of the calculations. First, we rewrite equation (2.4), such that

$$C(u) = \frac{w(u)C(u)}{w(u)} = \frac{A(u)}{w(u)},$$

where $A(u)$ is the numerator of (2.5). For the exact definition of A, its derivative, as well as w and the global algorithm, please refer to [PT97]. Here, we just want to point out the simplicity of these computations. After calculations, we obtain:

$$C'(u) = \frac{A'(u) - w'(u)C(u)}{w(u)}.$$

Following the same reasoning, we can move to two dimensions. We use the simplified definition of a NURBS surface :

$$S(u,v) = \sum_{i=0}^{l-d_1-1} \sum_{j=0}^{m-d_2-1} R_{ij}(u,v) P_{ij}$$

where

$$R_{i,j} = \frac{B_i^{d_1}(u) B_j^{d_2}(v) w_{i,j}}{\sum_{k=0}^{l-d_1-1} \sum_{\ell=0}^{m-d_2-1} B_k^{d_1}(u) B_\ell^{d_2}(v) w_{k,\ell}}.$$

Using the same technique, we used for the NURBS curve, we rewrite a NURBS surface as follows

$$S(u,v) = \frac{w(u,v)S(u,v)}{w(u,v)} = \frac{A(u,v)}{w(u,v)}.$$

Therefore we obtain the first derivative on u (symmetrically on v) as

$$\partial_u S = \frac{A_u(u,v) - w_u(u,v)S(u,v)}{w(u,v)}.$$

Nevertheless, we should know that as useful as these last two families of splines are, they require a Cartesian mesh to be defined in 2D. However, a big part of our work will be done in a hexagonal mesh that is not obtained by a convolution of two 1D vectors. Thus, we need an even more generalized form of splines.

Box-Splines, a less known spline family

Box-splines are a generalization of the well-known B-splines. They are also piecewise polynomials 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. As we mentioned in the Introduction, Chapter 1, a part of our study includes working on a hexagonal mesh, thus we need a basis that is adapted to a grid that cannot be obtained by a tensor product. There are mainly two families of splines that take advantage of this type of grid: hex-splines, first introduced in [VDVBU+04], and the three directional Box-splines [DBHR93, BHS13, DL91]. For a detailed comparison between these two types of splines we will refer to [CVDV07]. Based on the latter, we chose to use Box-splines, as the results are more stable.

General definition

A Box-spline is a spline that is more general than a NURBS and a B-spline. Nevertheless, despite being more flexible to different meshes, they remain less used

than B-splines or NURBS. This is due to their computation being slower and, in some cases, noisy. In fact, in general, it is better to use a type of splines that is adapted to the mesh used. However, for the hexagonal mesh, this generalization is mandatory. The general definition is [DBHR93, CVDV06]

Definition 6: 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 $\chi_\Xi \colon \mathbb{R}^d \longrightarrow \mathbb{R}$. 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.

$$\chi_\Xi(\mathbf{x}) = \begin{cases} \dfrac{1}{|\det(\Xi)|} & \text{if } \Xi^{-1}\mathbf{x} \in [0,1[^2, \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

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

$$\chi_{\Xi\cup\mathbf{v}}(\mathbf{x}) = \int_0^1 \chi_\Xi(\mathbf{x} - t\mathbf{v})\, dt. \tag{2.9}$$

Remark 2. *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 multiplicities of* e_1 *and* e_2 *are both* $d+1$*. See Appendix A for the proof.*

Remark 3. *The Box-splines can have different degrees in each direction. Thus, there are different definitions of the degree. We consider the definition below, which is specific to the kind of Box-splines we use in this paper.*

Definition 7: Three-directional Box-spline of degree d

Let Ξ be a 2×3 matrix with non-null columns in $\mathbb{R}^2$ such that they form a generating set of $\mathbb{R}^2$. Then, the three-directional Box-spline of degree d of generating matrix Ξ, χ^d_Ξ, is the Box-spline associated to Ξ, where all three generating vectors have multiplicity d.

In Figure 2.5, the difference between the first two Box-splines, $\chi_{[r_1,r_2]}$ and $\chi_{[r_1,r_2,r_3]}$ lies on the generating matrix. This is obvious when viewing the support of the box-splines in Figure 2.6. We notice that only $\chi_{[r_1,r_2,r_3]}$ and $\chi^2_{[r_1,r_2,r_3]}$

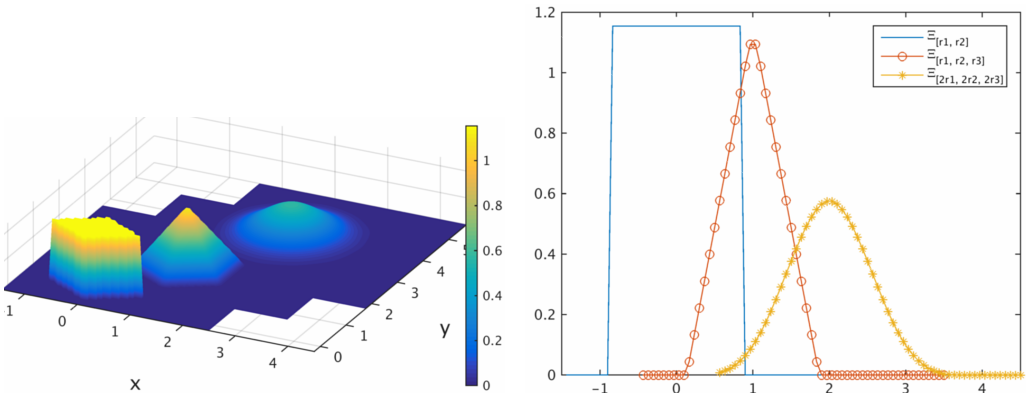


Figure 2.5: On the left: Box-splines $\chi_{[\mathbf{r}_1,\mathbf{r}_2]}$, $\chi_{[\mathbf{r}_1,\mathbf{r}_2,\mathbf{r}_3]}$ (shifted for better visualization), and $\chi^2_{[\mathbf{r}_1,\mathbf{r}_2,\mathbf{r}_3]}$ (also translated). On the right: 2d projections of the Box-splines onto the x plane.

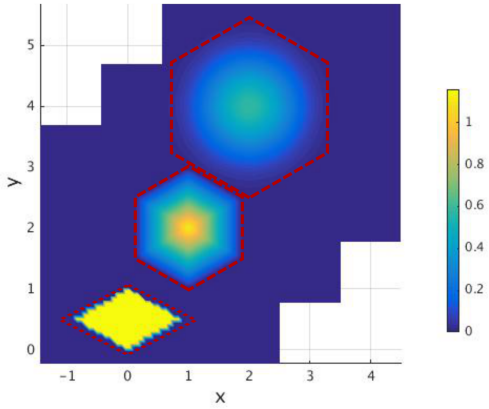


Figure 2.6: Support of Box-splines $\chi_{[\mathbf{r}_1,\mathbf{r}_2]}$, $\chi_{[\mathbf{r}_1,\mathbf{r}_2,\mathbf{r}_3]}$ (shifted for better visualization), and $\chi^2_{[\mathbf{r}_1,\mathbf{r}_2,\mathbf{r}_3]}$ (also translated).

are adapted to the hexagonal mesh and have a hexagonal support. In fact, these are the three-directional Box-splines of degree 1 and 2 in the hexagonal-mesh that will be mentioned further in the following sections.

Box-splines on the hexagonal mesh

We will use the Box-splines exclusively on the hexagonal mesh. For example, the first two Box-splines of Figures 2.5 and 2.6. We can use the definition of Condat and Van De Ville [CVDV06] for the Box-splines on a hexagonal mesh,

$$\chi^d(x,y) = \sum_{k_1,k_2=-d}^{d} \sum_{i=\max(k_1,k_2,0)}^{\min(d+k_1,d+k_2,d)} (-1)^{k_1+k_2+i} \binom{d}{i-k_1}\binom{d}{i-k_2}\binom{d}{i} \times \sum_{j=0}^{d-1} \binom{d-1+j}{j} \frac{1}{(2d-1+j)!\ (d-1-j)!} \times \left|\frac{2y}{\sqrt{3}} + k_1 - k_2\right|^{d-1-j} \left(x - \frac{k_1+k_2}{2} - \left|\frac{y}{\sqrt{3}} + \frac{k_1-k_2}{2}\right|\right)_+^{2d-1+j}. \tag{2.10}$$

We also want to remind the optimized algorithm for $d = 2$.

Algorithm 1: (`BoxSplineValue`) Computation of Box-splines of degree 2

Data: x, y, Cartesian coordinates of the point on which we evaluate the spline.
degree, the degree of the spline, here degree $= 2$.
Result: value, value of the Box-splines at given point (x, y)
Initialize u $= |x| - |y|/\sqrt{3}$
Initialize v $= |x| + |y|/\sqrt{3}$
if u ≤ 0 **then** u $=$ − u v $=$ v + u; `/* Symmetry r_2 */`
if u $\leq$ v / 2 **then** u $=$v − u; `/* Symmetry r_2+r_3 */`
Initialize g $=$ u − v / 2;
if v > 2 **then**
  value $= 0$
else if v < 1 **then**
  value $= 0.5 +$ ((5/3 − v/8)*v − 3)*v **2/4 + ((1 − v/4)*v + g **2/6 − 1)*g **2
else if u > 1 **then**
  value $=$ (v − 2.)**3*(g − 1.)/6.
else
  value $= 5/6 +$ ((1 + (1/3 − v/8)*v)*v/4 − 1)*v + ((1 − v/4)*v + g **2/6 − 1)*g **2
end

Mass and stiffness matrices

When solving PDE, for example even a simple Poisson equation, we come across some weak formulations that involve the Mass matrix and/or the Stiffness matrix (see equation (2.11)). The computation of these matrices is costly and generally not obvious as they require many terms and derivations of the basis functions. The computation of these matrices is well known when working in Cartesian meshes

or when working with B-splines. However, no previous work has been done to compute these matrices with Box-splines. This is mainly due to the fact that their description is too general. In this manuscript we specialize only in the three-directional Box-splines defined in Definition 7, thus we are able to compute and simplify the value of these matrices for the hexagonal mesh. The Mass and Stiffness matrices general definition is

$$\begin{aligned} M_{ij} &= \int \psi_i \, \psi_j \\ \Sigma_{ij} &= \int \nabla\psi_i \cdot \nabla\psi_j \end{aligned} \tag{2.11}$$

where ψ_i (respectively ψ_j) is the basis functions centered on the point of global index i (resp. j). Here the basis functions are actually the Box-splines $\chi^d_{\mathbf{R}}$. For more readability, we simplify this notation to χ^d for the following. While doing these computations, we realized that the easiest way to simplify these equations is by using the hexagonal coordinates[4] which are, for a point of global index i, $(k_1^i, k_2^i) = (m, n)$. Respectively, for a point of global index j, we use the hexagonal coordinates $(k_1^j, k_2^j) = (k, l)$. We denote by $\chi_i^d = \chi^d_{m,n}$, the Box-spline of degree d, centered on the point of index i. Using the properties in Appendix A, we obtained

$$\begin{aligned} M^d_{ij} &= \frac{2L^2}{N_c^2\sqrt{3}} \, \chi^{2d}_{m-k,n-l}(2d\,\mathbf{r}_3) \\ \Sigma^d_{ij} &= -\frac{2}{\sqrt{3}}\Big(\partial^2_{\mathbf{r}_1}\chi^{2d}_{m-k,n-l}(2d\,\mathbf{r}_3) + \partial^2_{\mathbf{r}_2}\chi^{2d}_{m-k,n-l}(2d\,\mathbf{r}_3) - \partial_{\mathbf{r}_1,\mathbf{r}_2}\chi^{2d}_{m-k,n-l}(2d\,\mathbf{r}_3)\Big). \end{aligned} \tag{2.12}$$

We remind the reader that L/N_c is the mesh step, and $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$ are the vectors generating the hexagonal mesh (see Equation (2.1)). Thus we can see that when solving a PDE using Box-splines of degree d, we will actually also need the values of the Box-splines with a degree 2 times higher, as well as their derivatives along the mesh vectors.

In Tables 2.1, 2.2 and 2.3, we gathered the results for Box-splines of degree 1 and 2 (for higher degrees and a python script to compute these results see Appendix A). To know the values of M^d_{ij} or Σ^d_{ij}, defined for Box-splines $\chi^d_i = \chi^d_{m,n}$ and $\chi^d_j = \chi^d_{k,l}$, we should read the value at the position $(c_1, c_2) = (2d + (m-k)N_c/L,\ 2d + (n-l)N_c/L)$. Notice that to find the values M^d_{ij} and Σ^d_{ij}, the values from the table should still be multiplied by $(2L^2)/(N_c^2\sqrt{3})$ for the Mass

[4] Unfortunately, this also introduces several new indexations making the equation even more difficult to understand on a first approach, but we tried to simplify them as much as possible. Hopefully, once the reader gets used to these notations, the lecture will become almost intuitive.

Mass matrix coefficients $\div \frac{2L^2}{N_c^2\sqrt{3}}$

c_1 \ c_2	1	2	3
1	$\frac{1}{12}$	$\frac{1}{12}$	0
2	$\frac{1}{12}$	$\frac{1}{2}$	$\frac{1}{12}$
3	0	$\frac{1}{12}$	$\frac{1}{12}$

Stiffness matrix coefficients $\div \frac{2}{\sqrt{3}}$

c_1 \ c_2	1	2	3
1	0.5	0.5	0
2	0.5	5	0.5
3	0	0.5	0.5

Table 2.1: Non-null terms of the Mass and Stiffness matrices for box-splines of degree 1

Mass matrix coefficients $\div 181440 \frac{2L^2}{N_c^2\sqrt{3}}$

c_1 \ c_2	1	2	3	4	5	6	7
1	1	17	17	1	0	0	0
2	17	868	2550	868	17	0	0
3	17	2550	18871	18871	2550	17	0
4	1	868	18871	47496	18871	868	1
5	0	17	2550	18871	18871	2550	17
6	0	0	17	868	2550	868	17
7	0	0	0	1	17	17	1

Table 2.2: Non-null terms of the Mass matrix for box-splines of degree 2

matrix, and by $2/\sqrt{3}$ for the Stiffness matrix. These values are useful mainly for the validation of the Finite-Element codes.

Finally, we have all the main properties to solve PDE such as the Poisson equation using a Finite Element or a Finite Differences approach. Nevertheless, the Vlasov equations will be solved using the Semi-Lagrangian scheme, for which an interpolation method is needed. This interpolation is described in the next section.

Interpolation using splines

In this section, we present the interpolation methods needed to apply the Semi-Lagrangian scheme on the logical domains. In fact, every step of the PDE solving

Stiffness matrix coefficients $\div 20160 \frac{2}{\sqrt{3}}$

c_1 \ c_2	1	2	3	4	5	6	7
1	-1	-43	-43	-1	0	0	0
2	-43	108	-1218	108	-43	0	0
3	-43	-1218	8153	8153	-1218	-43	0
4	-1	108	8153	23800	8153	108	-1
5	0	-43	-1218	8153	8153	-1218	-43
6	0	0	-43	108	-1218	108	-43
7	0	0	0	-1	-43	-43	-1

Table 2.3: Non-null terms of the Stiffness matrix for box-splines of degree 2

is done on the logical domain. We use the cubic B-spline interpolation, which is useful for any scheme based on a Cartesian mesh. Furthermore, we will present a comparable method on the hexagonal mesh.

Cubic B-Spline interpolation

As we previously mentioned in Section 1, Remark 1, the magnitude of the error is greatly dependent on the accuracy of the interpolation method, and it has been shown that, for a Cartesian mesh, the cubic spline interpolation yields the most competitive results [BM08]. Thus, let us recall the principle of the piecewise cubic B-spline interpolation as presented in [DB78].

Firstly, let us see the theory in a 1D uniform mesh, as it is simpler to understand and the extrapolation to 2D is natural. Given a set of point x_i in $[a, b]$ such that $x_i = a + ih,\ i = 0, \ldots, N$ and $h = \frac{b-a}{N}$. The associated data f is known at the mesh points, $f(x_1), \ldots, f(x_n)$. Let us define a function $\tilde{f}$ such that on each interval $[x_i, x_{i+1}]$ we have $\tilde{f} \in \Pi \leq 3$, the set of polynomials of maximum degree 3, and $\tilde{f}(x_i) = f(x_i)$. The function $\tilde{f}$ is an approximation of f using the cubic splines B^3. For illustration purposes, we consider a B-spline which knots are the mesh nodes.

This simplifies greatly the computation of the splines which formula is below.

$$B_j^3(x) = \frac{1}{6}\begin{cases} \left(2 - \dfrac{|x - x_j|}{h}\right)^3 & \text{if } h \leq |x - x_j| < 2h, \\ 4 - 6\left(\dfrac{x - x_j}{h}\right)^2 + 3\left(\dfrac{|x - x_j|}{h}\right)^3 & \text{if } 0 \leq |x - x_j| < h, \\ 0 & \text{else.} \end{cases} \tag{2.13}$$

Since the number of knots needed for the cubic splines (*i.e.* $= N + 2$) is bigger than the number of mesh points, the boundary conditions need to be properly defined. Several alternatives exist, for example, assigning the two extra knots to the extremities of the interval. This deforms the shape of the splines but makes them interpolatory. Another solution is to use the Greville abscissae rather than the knots (which implies knowing the initial data function at these points), so that the number of unknowns matches the number of knowns [Far14].

Whichever solution is chosen, the approximation of the function f is (or is nearly, differing on the extremities) given by

$$\tilde{f}(x) = \sum_{j=0}^{N+1} c_j B_j(x)$$

where the coefficients c_j are obtained by solving a linear system since we know the values of the interpolation at the mesh nodes,

$$\tilde{f}(x_i) = \sum_{j=0}^{N} c_j B_j(x_i) = f(x_i). \tag{2.14}$$

The $(N+2) \times (N+2)$ system is $A\mathbf{c} = \mathbf{f}$, where the matrix A contains the values of the cubic splines, $\mathbf{c}$ is the vector of the c_j coefficients and respectively $\mathbf{f}$ of the $f(x_i)$ values. Since the only non-null values of the splines at the grid points are $B_j(x_j) = \dfrac{2}{3}$ and $B_j(x_{j+1}) = B_j(x_{j-1}) = \dfrac{1}{6}$, the matrix A is tri-diagonal. Actually, we get

$$A = \frac{1}{6}\begin{pmatrix} 1 & 4 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & 0 & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 4 & 1 \end{pmatrix}. \tag{2.15}$$

We recall the reader that this matrix depends on the interpolation points used, and on the boundary conditions.

Imposing Hermite boundary conditions

Suppose we wish to impose Hermite boundary conditions at both extremities of our domain. We get

$$f'(x_0) \simeq \tilde{f}'(x_0), \quad f'(x_N) \simeq \tilde{f}'(x_N). \tag{2.16}$$

We have $B'_{i\pm1}(x_i) = \pm 1/(2h)$ and $B'_j(x_i) = 0$. Then, Equation (2.16) becomes

$$\begin{cases} f'(x_0) & \simeq & \frac{1}{2h}c_1 - \frac{1}{2h}c_{-1} \\ f'(x_N) & \simeq & \frac{1}{2h}c_{N+1} - \frac{1}{2h}c_{N-1}. \end{cases} \tag{2.17}$$

We notice the introduction of two new coefficients, c_{-1} and c_{N+1}. Hence, the linear system $A\mathbf{c} = \mathbf{f}$ is actually $(N+3) \times (N+3)$.

Quasi-interpolation on the hexagonal mesh

We have chosen for the hexagonal mesh a local quasi-interpolation method using Box-splines. By locality [dB90] we mean that the quasi-interpolation on a given point depends only on the data in the neighborhood of that point. Previous work has been done on quasi-interpolations in three-directional meshes [Sab96, Sab02], as well as quasi-interpolations using Box-splines[LMS08]. Nevertheless, a quasi-interpolation scheme specific to our hexagonal mesh using Box-spline has only been studied for signal-processing applications [CVDV07]. In this section, we intend to apply this scheme such that we can use it for the Semi-Lagrangian method.

We write the problem in the following way: let us consider an initial sample on the grid $s[\mathbf{k}] = f(\mathbf{Rk})$, 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 $\tilde{f}(\mathbf{x}) = \sum c[\mathbf{k}]\chi^d(\mathbf{x} - \mathbf{Rk})$, where χ^d are the Box-splines of degree d of matrix $\Xi = [\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}]$ and $c[\mathbf{k}]$ are the associated coefficients. These coefficients are defined such that $\tilde{f}(\mathbf{x})$ approximates $f(\mathbf{x})$ to a certain order $M = 2d$ or, in other words, the approximation is exact only if $f(\mathbf{x})$ is a polynomial of degree $M-1$ or less [CVDV07]. 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-spline 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 $\mathbf{x}_j$ of hexagonal coordinates $\mathbf{k}_j$ (*i.e.* such that $\mathbf{x}_j = \mathbf{Rk}_j$) by discrete

filtering [CVDVU06]. This filtering ensures the locality of the scheme, and is written such that

$$c[\mathbf{k}_j] = \sum_{\ell=0}^{K-1} s[\mathbf{k}_{V_{j,\ell}}]p[\ell] = \sum_{\ell=0}^{K-1} s[\mathbf{k}_\ell + \mathbf{k}_j]p[\ell], \tag{2.18}$$

where s is the initial sample data and $p[\ell]$ are K pre-filters which will be defined later on. V_j is the set of global indices of the K neighboring points of $\mathbf{P}_j$, thus $V_{j,\ell}$ is the global index of the ℓ-th point in our stencil, and $\mathbf{k}_{V_{j,\ell}}$ its hexagonal coordinates. We notice that $(\mathbf{k}_\ell + \mathbf{k}_j)$ also designates the same hexagonal coordinates. This is due to the properties of the hexagonal mesh, and the notations we chose in Section 2.

Box-spline coefficients

In [BU99], the authors show that a quasi-interpolation of order M is achieved when the following proposition, in Fourier form, is satisfied.

$$\hat{p}(\boldsymbol{\omega}) + \hat{\chi}^d(\boldsymbol{\omega} + 2\pi\,\hat{\mathbf{R}}\mathbf{k}) = \delta_\mathbf{k} + O(\|\boldsymbol{\omega}\|^M), \quad \forall \mathbf{k} \in \mathbb{Z}^2 \tag{2.19}$$

Based on the literature available, notably [CVDV07], we have chosen for Box-splines of degree 1 (*i.e.* $d = 1$) the quasi-interpolation pre-filters p_{IIR2} which seem to give better results within a competitive time. The pre-filter $p_{IIR2}[\ell]$ of the point of local index ℓ, for splines of degree 1, is defined by

$$p_{IIR2}[\ell] = \begin{cases} 1775/2304, & \text{if } \ell = 0, \\ 253/6912, & \text{if } 1 \leq \ell \leq 6, \\ 1/13824, & \text{if } 7 \leq \ell \leq 18 \text{ and } \ell \text{ odd}, \\ 11/6912, & \text{if } 7 \leq \ell \leq 18 \text{ and } \ell \text{ even}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.20}$$

Here $K = 19$, and for Box-splines of degree 2, $K = 61$. For higher degrees, we refer to the previously mentioned papers (particularly [CVDVU06]). To get the Box-spline coefficients, we use Equation (2.18), where $p[\ell] = p_{IIR2}[\ell]$.

Optimizing the evaluation

We have all the elements for the approximation of a function f with Box-splines of degree d

$$\tilde{f}(\mathbf{x}) = \sum_{\mathbf{k}\in\mathbb{Z}^2} c[\mathbf{k}]\chi^d(\mathbf{x} - \mathbf{R}\mathbf{k}). \tag{2.21}$$

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^d are only non-zeros in a limited number of points. Therefore we need to know the indices $\mathbf{k}$ such that $\chi^d(\mathbf{x}-\mathbf{Rk}) \neq 0$. For this purpose we will use the strategy suggested in [CVDV07]: to start we need to obtain the indices on the coordinate system generated by $\mathbf{R}$: $\tilde{\mathbf{k}} = [\lfloor u \rfloor \ \lfloor v \rfloor]$ where $(u\ v)^T = \mathbf{R}^{-1}\mathbf{x}$. Thus, for example, in the case $d = 1$ (see Figure 2.7), we only need four terms associated with the vertices of the encapsulating rhomboid (in grey): $\mathbf{R}\tilde{\mathbf{k}}$ (in blue), $\mathbf{R}\tilde{\mathbf{k}}+\mathbf{r_1}$ (in yellow), $\mathbf{R}\tilde{\mathbf{k}}+\mathbf{r_2}$ (in green) and $\mathbf{R}\tilde{\mathbf{k}}+\mathbf{r_1}+\mathbf{r_2}$ (in orange).

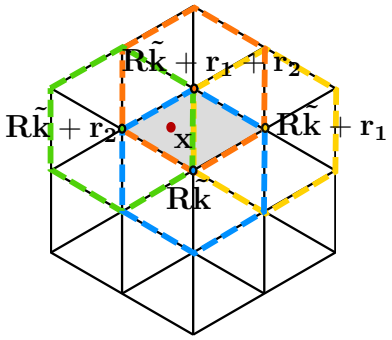


Figure 2.7: Representation of a point $\mathbf{x}$ (red) in a hex-mesh of $N_c = 2$. The rhomboid (grey) is formed by the intersection of all the possible non-null box-splines of degree $d = 1$ at the given point

Finally, we obtain:

$$\begin{aligned}\tilde{f}(\mathbf{x}) = \quad & c[\tilde{\mathbf{k}}]\ \chi^1(\mathbf{x}-\mathbf{R}\tilde{\mathbf{k}}) \\ & + c[\tilde{\mathbf{k}}+[1,0]]\ \chi^1(\mathbf{x}-\mathbf{R}\tilde{\mathbf{k}}-\mathbf{r_1}) \\ & + c[\tilde{\mathbf{k}}+[0,1]]\ \chi^1(\mathbf{x}-\mathbf{R}\tilde{\mathbf{k}}-\mathbf{r_2}) \\ & + c[\tilde{\mathbf{k}}+[1,1]]\ \chi^1(\mathbf{x}-\mathbf{R}\tilde{\mathbf{k}}-\mathbf{r_1}-\mathbf{r_2}). \end{aligned} \tag{2.22}$$

Remark 4. *The χ^1 spline has a support of radius the unity, thus one of the elements of (2.22) 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 5. *For the Box-splines of degree 2, in equation (2.22) there would be 16 coefficients to compute (see Figure 2.7) from which 4 would be null terms.*

Finally, we have all the main components for the resolution of the Vlasov equations both on a patch based domain and a hexagonal domain. Both approaches

use the concept of a coordinate transformation even though the first tests will be done on the logical domains. Firstly, we will see the most conventional approach, the Multi-patch Approach.

3

The Multi-patch Approach.

As shown in the introduction, in GYSELA, the current discretization of the tokamak's poloidal plane is far from satisfactory. The main reason is that, to avoid the singular point of the polar mesh, GYSELA uses a polar mesh with a hole in the middle (See Figure 3.1a).

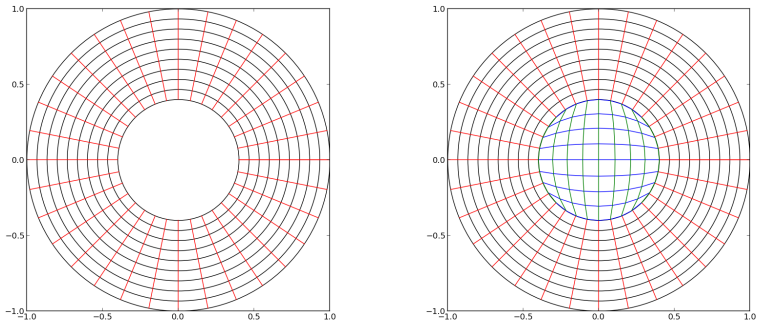
The first idea to solve this problem was to fill the void zone with another mesh that does not present a singularity in the center. Thus, having the global geometry decomposed in at least two patches. We defined this geometry in Chapter 3, while in Chapter 3 we discuss how to adapt the Semi-Lagrangian method to a Multi-patch circular configuration and to more complex geometries as well (*i.e.* a D-shape geometry). The adaptation relies on the IGA approach described in the previous chapter. We present the final algorithm, and the tools needed to define it. Finally, we present some results.

General concept: patch decomposition

The objective of this manuscript, described simply, is to find an alternative discretization to the current 2D poloidal cut in GYSELA without a singular point at the center and that is adaptable to more complex geometries. To meet the first criteria and be able to provide a method that requires minimal modifications to the code, we choose to keep the polar mesh, and introduce a second mesh (or patch) connected to it. We called this method, the Multi-patch Approach. For the second criteria, we decide to use the IGA approach, so that the generalization to any geometry is automatic.

Domain decomposition

The first approach means a transition from a polar annulus (See Figure 3.1a) to a domain where an internal disk with no geometrical singularities is coupled, in a $\mathcal{C}^1$ fashion, to a field aligned polar mesh (See Figure 3.1b).



(a) **(b)**

Figure 3.1: Gysela's current mesh for the poloidal plane and a new discretization

In Chapter 2 we mentioned that these meshes are generated from a coordinate transformation of a Cartesian mesh to the given physical grid. The mapping to form the external annulus is the well-known polar transformation.

$$F_{ext}(\eta, \xi) = (\eta \cos(\xi),\ \eta \sin(\xi))$$
$$F_{ext}^{-1}(x, y) = (\sqrt{x^2 + y^2},\ \arctan(y/x)).$$

Its associated Jacobian matrix is

$$J(F_{ext}) = \begin{pmatrix} \cos(\xi) & -\eta \sin(\xi) \\ \sin(\xi) & \eta \cos(\xi) \end{pmatrix}.$$

It is easy to notice that the determinant of the Jacobian is null at the origin. Thus, we encounter the expected numerical errors: division by zero at the origin and, when the mesh is fine, by numerical zeros at points close to the origin; this point is also geometrically singular, *i.e.* the cells near the origin become triangular.

To solve this problem, we define a new mesh for the internal disk; this mapping is $\mathcal{G}^1$ continuous with the polar annulus and contains no geometrical singular point (See Figure 3.1b). It is important to notice that this grid is not field-aligned and thus, in our context, we cannot discretize the whole domain with it. Nevertheless, it is still a regular mesh, which is important when using the Semi-Lagrangian

scheme. Indeed, for each time step, we locate the origin of the characteristics, so this procedure needs to be as light as possible. The internal mapping is as follows.

$$F_{int}(\eta,\xi) = \left((2\xi-1)\sqrt{1-\frac{(2\eta-1)^2}{2}},\ (2\eta-1)\sqrt{1-\frac{(2\xi-1)^2}{2}} \right). \tag{3.1}$$

Singular points

Geometrically this mapping has no singular points. This provides the advantage of choosing a coarser refinement at the origin of the domain.[1] Nevertheless, the four corners of the square patch $\mathcal{P} = [0,1]^2$ represent, numerically, a singularity. First of all, we introduce the function $F_{int,s}$, a simplified version of F_{int}, such that $F_{int,s}(2\eta-1,\ 2\xi-1) = F_{int}$. The transformation is written in the form

$$F_{int,s}(\eta,\xi) = \left(\xi\sqrt{1-\frac{\eta^2}{2}},\ \eta\sqrt{1-\frac{\xi^2}{2}} \right).$$

It contains the same number of singular points, only translated. Let us compute the Jacobian matrix of this mapping to study the four singular points.

$$J(F_{int,s}) = \begin{pmatrix} -\frac{\xi\eta}{2}\left(1-\frac{\eta^2}{2}\right)^{-1/2} & \sqrt{1-\frac{\eta^2}{2}} \\ \sqrt{1-\frac{\xi^2}{2}} & -\frac{\xi\eta}{2}\left(1-\frac{\xi^2}{2}\right)^{-1/2} \end{pmatrix}.$$

We compute its determinant

$$\begin{aligned} \det(J_{int,s}) &= \frac{\eta^2\xi^2}{4}\left(1-\frac{\eta^2}{2}\right)^{-1/2}\left(1-\frac{\xi^2}{2}\right)^{-1/2} - \left(1-\frac{\eta^2}{2}\right)^{1/2}\left(1-\frac{\xi^2}{2}\right)^{1/2} \\ &= \frac{\eta^2+\xi^2-2}{2\left(1-\frac{\eta^2}{2}\right)^{1/2}\left(1-\frac{\xi^2}{2}\right)^{1/2}}. \end{aligned}$$

The singular points are the points (η,ξ) such that $\det(J_{int,s}) = 0$. Thus, the zeros of the equation $\eta^2+\xi^2-2=0$, which is the equation of a paraboloid as seen in Figure 3.2. In these singular points the inverse mapping of F does not exist.

[1] This is an important property since, generally, turbulence is not expected to generate small scales near the magnetic axis. Thus, at the origin of the domain, the mesh can be coarser.

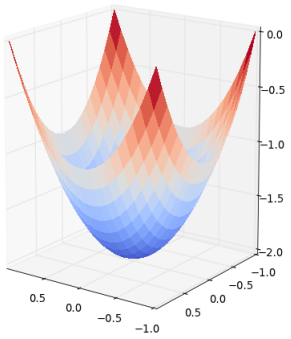


Figure 3.2: Paraboloid defined by equation $\eta^2 + \xi^2 - 2$

We notice that if $\eta = \pm 1$ and $\xi = \pm 1$ the Jacobian's determinant is null. Therefore, there are four singular points, numerically, of values:

$$P_1^s = (-1, -1),\ P_2^s = (-1, 1),\ P_3^s = (1, -1),\ P_4^s = (1, 1).$$

The singular points of this mapping will be the four "corners" of the patch. Nevertheless, as these four points are singular numerically and not geometrically, we strongly believe we can handle the repercussions it may have on the results. We can thus continue with the construction of the domain.

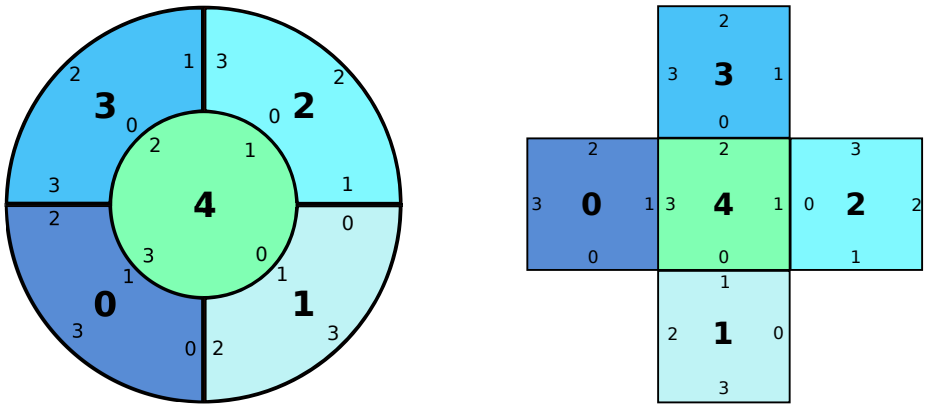
Modeling the domain using CAID

We are going to take advantage of the external crown symmetry to subdivide it into four identical patches with a given rotation from the origin. As we can see in Figure 3.3a, we obtain five patches, which have at the most one edge in common with one another. The computation of the boundary conditions will be therefore easier.

As we mentioned earlier, the Semi-Lagrangian method is applied in the patch representation. However, in the code, every patch is independent of its geometrical representation, so we need to have a notation system to index every patch, every edge, and the relationship between each patch (neighbors, connecting edges, etc.). Figures 3.3a and 3.3b show the notations used in the physical and logical domains.

To create this geometry we use the software CAID[2]. The resulting geometry is an object containing, between other attributes, the information about the number of patches, the connectivity between them (see below), and their respective coordinate transformation and derivatives. These three elements are all we need to define our new scheme and domain.

[2]Created by A. Ratnani, CAID is a "multi-platform software for Isogeometric Analysis pre- and post-Processing. Its design goal is to provide a fast, light, user-friendly, and meshing tool." http://ratnani.org/caid_doc/index.html



(a) Patch subdivision of domain **(b)** Patch representation of connectivity

Figure 3.3: Patch configuration of a disk using CAID

To symbolize the relationships between patches we use two connectivity lists. Each list element of index i is a couple of integers. The first integer will represent the patch number and the second, the edge number. Moreover, the couple (p_i, b_i) of index i in any of the lists is associated to the couple (p'_i, b'_i) of index i in the other list. We will denote this connectivity by $(p_i, b_i) \equiv (p'_i, b'_i)$. For the previously mentioned example we obtain the lists:

```
list_conn1 = [ [0,0], [0,1], [0,2], [1,0],
               [1,1], [2,0], [2,3], [3,0] ]
list_conn2 = [ [1,2], [4,3], [3,3], [2,1],
               [4,0], [4,1], [3,1], [4,2] ]
```

Multi-patch Semi-Lagrangian method

We defined a Multi-patch geometry and we set a way to describe the connectivity between patches. Nonetheless, we still need to determine the tools needed when adapting the Semi-Lagrangian method to such a geometry. We present the principle using a simple model, the advection equation, which requires already most of the tools needed for more complex models.

Solving the constant advection equation

As a first example, we study the application of the Semi-Lagrangian method to the well-known constant coefficient advection equation on a Multi-patch domain.

Similar to what we did in Chapter 1. This reminder will help us transition to the IGA approach. Firstly, we need to solve the model in Cartesian coordinates. Indeed, we estimate the error of the numerical solution by comparing it to the exact solution. Let $\mathbf{x} = (x, y)$, the model is defined by

$$\frac{\partial f}{\partial t}(t, x, y) + \mathbf{A} \cdot \nabla_{\mathbf{x}} f(t, x, y) = 0 \qquad \mathbf{x} \text{ in } \Omega, \tag{3.2}$$

with initial condition

$$f(0, x, y) = f_0(x, y) \tag{3.3}$$

where $\mathbf{A}$ is divergence free, *i.e.* $\nabla_{\mathbf{x}} \cdot \mathbf{A} = 0$, more precisely $\mathbf{A}$ is the constant advection coefficient so that $\mathbf{A} = (a_1, a_2)^T$. Using the method of characteristics, we find the solution of Equation (3.2) in the physical space. To find the characteristics we solve the following system of ODE.

$$\begin{cases} \dfrac{\partial \mathbf{X}}{\partial t} &= \mathbf{A} \\ \mathbf{X}(s) &= \mathbf{x}. \end{cases}$$

We should note that even if the domain is subdivided in patches, it does not change anything to the model, nor to the solution which is

$$\begin{cases} X(t) &= x + a_1 \cdot t \\ Y(t) &= y + a_2 \cdot t \end{cases}$$

Knowing the initial condition $f(0;\ x, y) = f_0(x, y)$, we obtain

$$f(t, x, y) = f_0(X(t), Y(t)) = f_0(x - a_1\, t,\ y - a_2\, t). \tag{3.4}$$

This model is written in Cartesian coordinates. The solution is easy to compute and will be useful to estimate the accuracy and approximation of the method. However, the principle of the IGA approach is that the model is solved on the logical space. Thus, we need to write it in a more general form.

Changing the coordinate system

We wish to write Equation (3.2) in the logical space (we note that Equation (3.3) is automatically transferable to the logical space) which coordinate system is denoted by $\boldsymbol{\eta} = (\eta, \xi)$. Let us introduce the mapping from the logic to the physical coordinates, and its inverse.

$$F(\boldsymbol{\eta}) = \mathbf{x} \Leftrightarrow \begin{cases} x &= F_1(\eta,\xi) \\ y &= F_2(\eta,\xi) \end{cases} \quad \text{and} \quad F^{-1}(\mathbf{x}) = \boldsymbol{\eta} \Leftrightarrow \begin{cases} \eta &= F_1^{-1}(x,y) \\ \xi &= F_2^{-1}(x,y) \end{cases}.$$

Their associated Jacobian matrices are respectively

$$\mathbf{J}(F) = \mathbf{J} = \begin{pmatrix} \dfrac{\partial F_1}{\partial \eta} & \dfrac{\partial F_1}{\partial \xi} \\ \dfrac{\partial F_2}{\partial \eta} & \dfrac{\partial F_2}{\partial \xi} \end{pmatrix}, \quad \mathbf{J}(F^{-1}) = \mathbf{J}^{-1} = \frac{1}{|\mathbf{J}|} \begin{pmatrix} \dfrac{\partial F_2}{\partial \xi} & -\dfrac{\partial F_1}{\partial \xi} \\ -\dfrac{\partial F_2}{\partial \eta} & \dfrac{\partial F_1}{\partial \eta} \end{pmatrix} \tag{3.5}$$

where $|\mathbf{J}|$ is the determinant of $\mathbf{J}$. We suppose here that $F(\boldsymbol{\eta})$ follows the Inverse Function theorem, which states that a function is invertible if $|\mathbf{J}| \neq 0$. Thus we can write Equation (3.5). We wish now to write the gradient of f in the patch coordinate system using the mapping. Thus, we apply the chain rule for 2 coordinates to f, we obtain

$$\begin{cases} \dfrac{\partial f}{\partial x} &= \dfrac{\partial f}{\partial \eta}\dfrac{\partial F_1^{-1}}{\partial x} + \dfrac{\partial f}{\partial \xi}\dfrac{\partial F_2^{-1}}{\partial x} \\ \dfrac{\partial f}{\partial y} &= \dfrac{\partial f}{\partial \eta}\dfrac{\partial F_1^{-1}}{\partial y} + \dfrac{\partial f}{\partial \xi}\dfrac{\partial F_2^{-1}}{\partial y} \end{cases}.$$

Then we can rewrite the gradient of f,

$$\nabla_{\mathbf{x}} f(x,y) = \begin{pmatrix} \partial_x f \\ \partial_y f \end{pmatrix} = \begin{pmatrix} \dfrac{\partial f}{\partial \eta}\dfrac{\partial F_1^{-1}}{\partial x} + \dfrac{\partial f}{\partial \xi}\dfrac{\partial F_2^{-1}}{\partial x} \\ \dfrac{\partial f}{\partial \eta}\dfrac{\partial F_1^{-1}}{\partial y} + \dfrac{\partial f}{\partial \xi}\dfrac{\partial F_2^{-1}}{\partial y} \end{pmatrix}.$$

We obtain

$$\nabla_{\mathbf{x}} f = \mathbf{J}^{-T} \nabla_{\boldsymbol{\eta}} \tilde{f}(\eta,\xi) \tag{3.6}$$

with $\mathbf{J}^{-T} = \mathbf{J}(F^{-1})^T$ and $\tilde{f}(F(\eta,\xi)) = f(x,y)$. Moreover, we need to transform the advection vector to the new domain. Let us notice that, if we define a function ψ so that $\psi = a_1 y - a_2 x$ we can rewrite the advection vector as a curl.

$$\overrightarrow{rot}_{\mathbf{x}}\, \psi = \begin{pmatrix} \partial_y \psi \\ -\partial_x \psi \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}. \tag{3.7}$$

Using this notation (3.2) becomes

$$\frac{\partial f}{\partial t}(x,y) + \overrightarrow{rot}_{\mathbf{x}}\, \psi \cdot \nabla_{\mathbf{x}} f(x,y) = 0. \tag{3.8}$$

Furthermore,

$$\psi(x,y) = a_1 y - a_2 x = a_1 F_2 - a_2 F_1 = \tilde{\psi}(\eta,\xi).$$

Let us compute the curl of $\tilde{\psi}$,

$$\overrightarrow{rot_{\boldsymbol{\eta}}}\,\tilde{\psi} = \begin{pmatrix} a_1\partial_\xi F_2 - a_2\partial_\xi F_1 \\ -a_1\partial_\eta F_2 + a_2\partial_\eta F_1 \end{pmatrix} = (\mathbf{J}(F))^{-1}\overrightarrow{rot_{\mathbf{x}}}\,\psi\,|\mathbf{J}|. \tag{3.9}$$

Thus,

$$\overrightarrow{rot_{\mathbf{x}}}\,\psi \;=\; \frac{1}{|\mathbf{J}|}\,\mathbf{J}\,\overrightarrow{rot_{\boldsymbol{\eta}}}\,\tilde{\psi}. \tag{3.10}$$

Using the results (3.6) and (3.10), we can write

$$\begin{aligned} & \frac{\partial \tilde{f}}{\partial t}(\eta,\xi) + \frac{1}{|\mathbf{J}|}\,\mathbf{J}\,\overrightarrow{rot_{\boldsymbol{\eta}}}\,\tilde{\psi}\cdot\mathbf{J}^{-T}\,\nabla_{\boldsymbol{\eta}}\,\tilde{f}(\eta,\xi) = 0 \\ \iff \quad & \frac{\partial \tilde{f}}{\partial t}(\eta,\xi) + \frac{1}{|\mathbf{J}|}\,\mathbf{J}\mathbf{J}^{-1}\,\overrightarrow{rot_{\boldsymbol{\eta}}}\,\tilde{\psi}\cdot\nabla_{\boldsymbol{\eta}}\,\tilde{f}(\eta,\xi) = 0 \\ \iff \quad & \frac{\partial \tilde{f}}{\partial t} + \frac{1}{|\mathbf{J}|}\,\overrightarrow{rot_{\boldsymbol{\eta}}}\,\tilde{\psi}\cdot\nabla_{\boldsymbol{\eta}}\,\tilde{f} = 0 \end{aligned} \tag{3.11}$$

Introducing the notation $\tilde{\mathbf{A}} = \dfrac{1}{|\mathbf{J}|}\overrightarrow{rot_{\boldsymbol{\eta}}}\,\tilde{\psi}$, we obtain

$$\frac{\partial \tilde{f}}{\partial t} + \tilde{\mathbf{A}}\cdot\nabla_{\boldsymbol{\eta}}\,\tilde{f} = 0.$$

This equation remains conservative as the advection coefficient is divergence free, *i.e.* $\nabla\cdot\tilde{\mathbf{A}} = 0$. We summarize the results in the definition below.

Definition 8: Advection equation on general coordinates

Given a coordinate transformation $F(\eta,\xi) = (F_1(\boldsymbol{\eta}), F_2(\boldsymbol{\eta})) = (x,y)$ of Jacobian $|\mathbf{J}|$, the constant coefficient advection equation in general coordinates reads

$$\frac{\partial \tilde{f}}{\partial t} + \tilde{\mathbf{A}}\cdot\nabla_{\boldsymbol{\eta}}\,\tilde{f} = 0 \tag{3.12}$$

where $\tilde{\mathbf{A}} = \dfrac{1}{|\mathbf{J}|}\overrightarrow{rot_{\boldsymbol{\eta}}}\,\tilde{\psi}$, $\tilde{\psi}(\eta,\xi) = a_1 F_2(\boldsymbol{\eta}) - a_2 F_1(\boldsymbol{\eta})$, and $\mathbf{A} = (a_1\ a_2)^T \in \mathbb{R}^2$.

With this definition of the model in logical space, we can apply the Semi-Lagrangian method to solve it.

Applying the classical Semi-Lagrangian method

Let us apply the method of characteristics to the constant advection equation written in general coordinates:

$$\frac{\partial \tilde{f}}{\partial t}(\eta,\xi) + \frac{1}{|\mathbf{J}|}\overrightarrow{rot_{\boldsymbol{\eta}}}\,\tilde{\psi}\cdot\nabla_{\boldsymbol{\eta}}\,\tilde{f} = 0. \tag{3.13}$$

We denote by $H(t;\boldsymbol{\eta},s)$ the model's characteristics. Their associated curves are the following ODE solution

$$\begin{cases} \dfrac{\partial \mathbf{H}}{\partial t} &= \dfrac{1}{|\mathbf{J}|}\overrightarrow{rot_{\boldsymbol{\eta}}}\,\tilde{\psi} &= \tilde{\mathbf{A}}(\eta,\xi) \\ \mathbf{H}(s) &= \boldsymbol{\eta}_0 \end{cases} \tag{3.14}$$

where

$$\overrightarrow{rot_{\boldsymbol{\eta}}}\,\tilde{\psi}(\eta,\xi) = \begin{pmatrix} a_1\partial_\xi F_2 - a_2\partial_\xi F_1 \\ -a_1\partial_\eta F_2 + a_2\partial_\eta F_1 \end{pmatrix}$$

The Semi-Lagrangian method is based on two steps to update the distribution function $\tilde{f}^{n+1}$ at t_{n+1} from its value $\tilde{f}^n$ at time t_n:

1. For each grid point $\boldsymbol{\eta}_i$ compute $\mathbf{H}(t_{n+1};\boldsymbol{\eta}_i,t_n)$ the value of the characteristic at t_{n+1} which takes the value $\boldsymbol{\eta}_i$ at the previous time step t_n. To compute their value, we solve the system Equation (3.14) with a Runge-Kutta solver of order 4. Since the advection coefficients depend on the transformation function, this step has to be done in each patch individually and cannot be generalized.

2. We know that the distribution function $\tilde{f}$ solution of equation (3.13) is so that: $\tilde{f}^{n+1}(\boldsymbol{\eta}_i) = \tilde{f}^n(\mathbf{H}(t_{n+1};\boldsymbol{\eta}_i,t_n))$. Thus, we know the value of $\tilde{f}^{n+1}$ by interpolating on $\mathbf{H}(t_{n+1};\boldsymbol{\eta}_i,t_n)$ which are generally not grid points.

The difference between the classical Semi-Lagrangian scheme and the new Multi-patch adaptation relies on the previous two points. In fact, for the first one, when back-tracing the mesh points we might find some that go from one patch to another. We will see this in the next session. As for the second point, the interpolation method depends heavily on the boundary conditions. And these boundary conditions depend themselves on the neighboring patches. We will also present this in a following section.

Back-tracing particles on a Multi-patch domain

Let us start with the back-tracing of the mesh points characteristics. We have already stated that generally the origin of the characteristic will not be a point of the grid. Nevertheless, most often the advection is small enough so that it stays on the original patch, as in Figure 3.4.

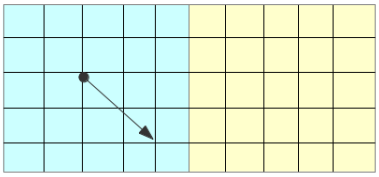


Figure 3.4: Tracing of characteristics: particle stays on the same patch

This is the ideal situation (the next step is a simple and straightforward interpolation algorithm, see Equation (2.10)), unfortunately, there are two other possible scenarios which require further analysis. The first one is still a common scenario when using the Semi-Lagrangian scheme. The second one, on the other hand, is specific to the Multi-patch Approach. The two scenarios are sketched in Figure 3.5.

(a) SLM - particle out of domain

(b) SLM - particle to neighbor patch

Figure 3.5: Special cases of the back-tracing of a particle for the SLM

Let us note $\mathcal{P}_\alpha$ and $\mathcal{P}_\beta$ two adjacent patches. At the time t_{n+1}, we can compute the characteristic $\mathbf{H}_{i,\alpha}^{n+1} = \mathbf{H}(t_n; \boldsymbol{\eta}_{i,\alpha}, t_{n+1})$ of the point $\boldsymbol{\eta}_i$ on the patch $\mathcal{P}_\alpha$, noted $\boldsymbol{\eta}_{i,\alpha}$ (equivalent to $\mathbf{X}_i^{n+1} = \mathbf{X}(t_n; \mathbf{x}_i, t_{n+1})$ in the physical coordinates) by solving a simple ODE specific to the model being solved. In general $\mathbf{H}_{i,\alpha}^{n+1} \in \mathcal{P}_\alpha$, meaning that the foot and the origin of the characteristics will be on the same patch, although this will not be always true. Verifying if the origin of the characteristic is on the same patch as its foot is almost automatic as by definition $\mathcal{P}_\alpha = \{(\eta, \xi), \quad 0 \leq \eta, \xi \leq 1\}$. Supposing $\mathbf{H}_{i,\alpha}^{n+1} \notin \mathcal{P}_\alpha$, several questions arise: is the origin of the characteristic on another patch? Or is it out of the domain? If it is in another patch, *e.g.* $\mathcal{P}_\beta$, how do we compute $\mathbf{H}_{i,\beta}^{n}$, its value on that patch?

If we are working on the 5-patch decomposition (Figure 3.3b), it is easy to know if a particle left the domain or if it went to another patch. Indeed, it is enough

to identify from which face the particle left the domain, and see the connectivity associated to that face. For example, from patch $\mathcal{P}_4$, the particle can never leave the domain. However, when looking for a general solution that works for any geometry chosen, the most obvious proceeding is the following.

$$\boldsymbol{\eta}_{i,\alpha} \xrightarrow[\mathbf{1}]{SLM} \mathbf{H}_{i,\alpha}^{n+1} \xrightarrow[\mathbf{2}]{F^{-1}} \mathbf{X}_{i,\beta}^{n+1} \xrightarrow[\mathbf{3}]{F} \mathbf{H}_{i,\beta}^{n+1}$$

where $\mathbf{H}_{i,\alpha}^{n+1}$ is the characteristics on the patch $\mathcal{P}_\alpha$. Transformations 1 and 3 have already been described and do not represent a high level of difficulty. On the other hand, the computation of F^{-1} is far more complicated (and even, at the singular points, nonexistent) [ALG+11]. For this reason, we will stick to simple compositions, where each patch boundary corresponds either to another patch edge or to a segment of the external boundary. In those conditions, the inter-patch advection can be done in the following fashion.

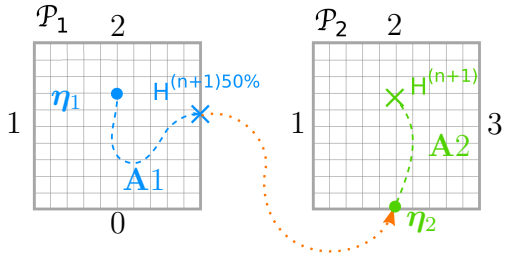


Figure 3.6: Sketch of an inter-patch advection with simplified notations

We have developed a small algorithm to handle inter-patch advections. It is based on the assumption that for a small advection (or time step) we can suppose that the trajectory can be simplified into a straight line. Let a configuration with two neighboring patches $\mathcal{P}_1$ and $\mathcal{P}_2$, with connectivity $(1,3) \equiv (2,0)$. For the remaining of this section, we have omitted the point indexation since it gives no additional information (*e.g.* $\boldsymbol{\eta}_{i,1}$ becomes $\boldsymbol{\eta}_1$). If for a given point $\boldsymbol{\eta}_1$, the origin of its characteristic $\mathbf{H}_1^{n+1}$ is outside $\mathcal{P}_1$, we execute the following five steps (sketched in Figure 3.6).

1. Compute the percentage of the given advection so that the characteristic remains in the domain. We denote this percentage $per = \mathbf{A}1/\mathbf{A}$ where $\mathbf{A}$ is the advection vector and $\mathbf{A}1$ is the advection done in the patch $\mathcal{P}_1$ and the intermediate point is noted $\mathbf{H}_1^{(n+1),per}$;

2. Find the face number of $\mathcal{P}_1$ on which $\mathbf{H}^{(n+1),per}$ lies (*e.g.* face 3 in the configuration on Figure 3.6). Get, from the connectivity lists, the associated patch and face number (*e.g.* patch $\mathcal{P}_2$ and face 0);

3. Compute the equivalent point of $\mathbf{H}_1^{(n+1),per}$ on its neighboring patch, we call this point $\boldsymbol{\eta}_2$;

4. Perform the remaining advection;

5. Check that if the obtained point is on the patch. If not, start over from point 1, else the final point is denoted $\mathbf{H}_2^{(n+1)}$.

Numerically, we added for each characteristic, the ID of the patch on which they lie. Indeed, the interpolation has to be done on the patch associated to the origin of the characteristic. This interpolation, which is the second main point that differs the classic BSL to our Multi-patch Semi-Lagrangian (MPSL) method, is a cubic spline interpolation[3] with Hermite boundary conditions.

Another possibility for $\mathcal{C}^0$ connectivity is to add Greville points in the first and last intervals. This solution is yet to be explored. Nonetheless, we believe the designed MPSL method is a solution to our problem. The novelty of this method is the way in which the boundary condition's slopes are computed. In fact, the latter depend on the neighboring patches (domain subdivision) and on the coordinate transformation (IGA approach). The following sections are dedicated on the novel aspects of the interpolation.

Interpolation boundary conditions

Since we are working on a Multi-patch structure, we need to pay special attention to the boundary conditions. Let us define by Ω the global domain formed by the union of the sub-domains (or patches) $\mathcal{P}_0, \ldots, \mathcal{P}_{N_{pat}-1}$. In what follows, $\partial\Omega$ denotes the domain's boundary (the union of all solid edges in Figure 3.7) and $\partial\mathcal{P}_i$ all boundaries of a given patch $\mathcal{P}_i, \forall i = 0, \ldots, N_{pat} - 1$. We introduce the notation $\partial\Gamma_{i,j}$, which represents the boundary between patches $\mathcal{P}_i$ and $\mathcal{P}_j$ (dashed lines in Figure 3.7).

Some properties follow:

- The union of all the patches boundaries is equal to the union of the external and all the internal boundaries *i.e.* $\bigcup_{i=0}^{N_{pat}-1} \partial\mathcal{P}_i = \bigcup_{i,j=0}^{N_{pat}-1} \partial\Gamma_{i,j} \cup \partial\Omega$

- The patch-to-patch boundary is symmetric, *i.e.* $\partial\Gamma_{i,j} = \partial\Gamma_{j,i}$

[3] The algorithm used for this interpolation is from the SeLaLib library, http://selalib.gforge.inria.fr/

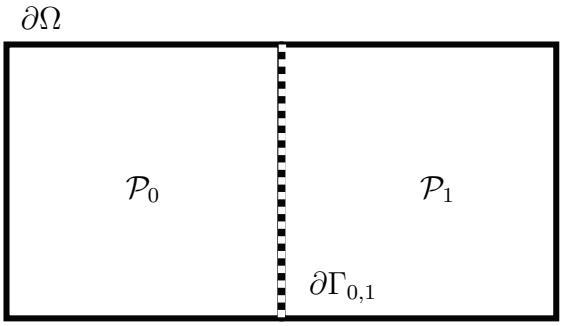


Figure 3.7: Boundaries definitions between a two-patch domain

When talking of boundary conditions, we will differentiate two types: we will refer to an "interior boundary condition" ($\partial\Gamma_{i,j}$), when considering a patch-to-patch edge, and to an "exterior boundary condition" ($\partial\Omega$), when is a domain edge.

For the exterior boundary conditions we generally use null Dirichlet boundary conditions or even periodic boundary conditions in special cases (*e.g.* external boundaries of Test-case 1). As for the internal boundaries, we impose Hermite boundary conditions which, written in Cartesian coordinates, give

$$\begin{cases} \dfrac{\partial f^{(i)}}{\partial x}(t,x,y) = \dfrac{\partial f^{(j)}}{\partial x}(t,x,y), & \forall (x,y) \in \partial\Gamma_{i,j}, \quad \text{if } y \text{ constant along } \partial\Gamma_{i,j}, \\ \dfrac{\partial f^{(i)}}{\partial y}(t,x,y) = \dfrac{\partial f^{(j)}}{\partial y}(t,x,y), & \forall (x,y) \in \partial\Gamma_{i,j}, \quad \text{else;} \end{cases} \tag{3.15}$$

where $f^{(i)}$ (respectively $f^{(j)}$) is the distribution function on $\mathcal{P}_i$ (resp. $\mathcal{P}_j$). The computation of these derivatives has to be of the right order of approximation. We detail the procedure used in the next section. Since the interpolations are done on the logical domains, the system (3.15) should be written on the logical coordinates. For clarity, we will explain the approximations of these slopes using the Cartesian coordinates.

Remark 6. *The type of boundary condition imposed for the interior boundaries was not natural. We tried several different types. From Dirichlet to Neumann passing by others that we have probably already forgotten. Unfortunately, since all this work was done towards the beginning of the thesis, it was not documented or the results were far from satisfactory. We do not consider any of the remaining results interesting enough to be presented in this manuscript.*

Derivatives approximations

To simplify this section, let $F(\boldsymbol{\eta}) = \mathrm{Id}(\boldsymbol{\eta}) = \mathbf{x}$. Since we only need the points coordinates, we use the more familiar Cartesian coordinates instead of the logical ones. However, the gradients are computed on the patches, and this simplification is only for the reader's ease. We will also detail the derivatives only along the first coordinate x.

So far, the algorithms used in our method are the cubic spline interpolation and the Runge-Kutta scheme of order 4 (to compute the origin of the characteristics). Thus, the method we choose to apply numerically the System (3.15) needs to be of the same order. For a long time, our code used high-order finite differences to approximate these derivatives. However, the results showed oscillations consistently around the interior boundary, and poor mass conservation. Until one day we came across the work of N. Crouseilles, *et al.* [CLS07]:

> Different ways have been explored to obtain the derivatives: finite differences of different orders, cubic spline approximation ... In order to reconstruct a smooth approximation (let us say $\mathcal{C}^1$ on the global domain), the cubic spline approximation has been chosen. Indeed, we remark that even in regions where f is smooth enough, a Finite Differences approximation remains quite different from a cubic spline approximation given by (5). Hence, as we want to reconstruct the distribution function via a cubic spline approximation, the first line of the linear system the matrix of which is given by (7) can introduce some numerical errors which can be propagated in the rest of the system; in the numerical experimentations we have driven, the final results are damaged, especially when one observes the mass conservation. Indeed, the Finite Differences approximation leads to some variations of the mass conservation which is an inconvenient for the long time behavior of the numerical solution. On the contrary, the approximation of the derivatives using cubic splines enables us to obtain a robust code with a relatively small number of discrete points.

After implementing the suggested cubic spline approximation for the slopes computation, the above-mentioned problems went away. Since their problem is not exactly equivalent to ours, the remaining of this section is dedicated to adapt their solution to our context.

First, let us introduce some notations, as similar as possible as the ones in [CLS07]. We suppose we are in a two-patch configuration, similar to the one on Figure 3.7, *i.e.* $\Omega = \mathcal{P}_1 \bigcup \mathcal{P}_2$. From here on, the numbering $\{i, j\}$ is reserved for the discretization. We intend to approximate the slopes along the x-direction on the internal boundary $\partial\Gamma_{1,2} \neq \varnothing$. We denote these slopes $s'(x) = \partial_x f^{(1)}(t, x, y) = \partial_x f^{(2)}(t, x, y)$ with $(x, y) \in \partial\Gamma_{1,2}$. Since in this context t and y are constant, we write $f^{(1)}(t, x, y) = f^{(1)}(x)$. We suppose the patch discretization is $\mathcal{C}^1$ continuous

and that the mesh step, h, is the same in both directions and patches. Thus, to get a notation analogue to [CLS07], we will use a continuous indexing for the points in the x-direction on both patches, see Figure 3.8. We suppose $x_i \in \partial\Gamma_{1,2}$, $\{x_{i-j}\} \in \mathcal{P}_1$ and $\{x_{i+j}\} \in \mathcal{P}_2$, $\forall j = 0, \cdots, N-1$. Hence, we do not need to specify on which patch the distribution function is. Actually, $f(x_{i+j})$ is in $\mathcal{P}_1$ if $j \leq 0$ and in $\mathcal{P}_2$ if $j \geq 0$. Lastly, we recall Equation (2.14) which gives the approximation of the distribution by cubic spline interpolations: $\tilde{f}(x_i) = \sum_{j=0}^{N} c_j B_j(x_i) = f(x_i)$ with B_j the j-th cubic spline and c_j the corresponding coefficient.

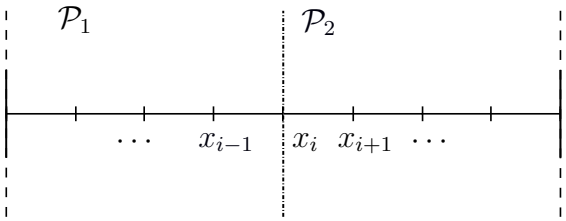


Figure 3.8: Temporary indexing to compute internal boundaries slopes

We proceed to approximate the slopes $s'(x_i)$. We start with the following two equations adapted from (2.17) and (2.15)

$$s'(x_i) = f'(x_i) \simeq \frac{1}{2h} c_{i+1} - \frac{1}{2h} c_{i-1} \tag{3.16}$$

$$f_i = \tilde{f}(x_i) = \frac{1}{6} c_{i-1} + \frac{2}{3} c_i + \frac{1}{6} c_{i+1}. \tag{3.17}$$

We notice (3.17) is a formula to obtain any given c_i

$$c_i = \frac{3}{2}\left(f_i - \frac{1}{6} c_{i-1} - \frac{1}{6} c_{i+1}\right) = \frac{3}{2} f_i - \frac{1}{4} c_{i-1} - \frac{1}{4} c_{i+1}. \tag{3.18}$$

By using (3.18) to compute c_{i+1} and c_{i-1} and injecting the results in (3.16) we obtain

$$\begin{aligned} s'(x_i) &= \frac{1}{2h}\left(\frac{3}{2} f_{i+1} - \frac{1}{4} c_i - \frac{1}{4} c_{i+2} - \frac{3}{2} f_{i-1} + \frac{1}{4} c_{i-2} + \frac{1}{4} c_i\right) \\ &= \frac{3}{4h}\left(f_{i+1} - f_{i-1}\right) - \frac{1}{8h}\left(c_{i+2} - c_{i-2}\right) \\ &= \frac{3}{4h}\left(f_{i+1} - f_{i-1}\right) - \frac{1}{4}\left(s'(x_{i-1}) + s'(x_{i+1})\right). \end{aligned} \tag{3.19}$$

This is already an approximation of the slope. However to get a higher order approximation, we use formula (3.19) to compute $s'(x_{i\pm1})$ and we inject the results back in (3.19) itself. After simplifications, the equation becomes

$$\begin{aligned} s'(x_i) &= \frac{3}{4h}\left(f_{i+1}-f_{i-1}\right) - \frac{3}{16h}\left(f_{i+2}-f_{i-2}\right) + \frac{1}{16}\left(s'(x_{i-2})+2s'(x_i)+s'(x_{i+2})\right) \\ \iff \frac{7}{8}s'(x_i) &= \frac{3}{4h}\left(f_{i+1}-f_{i-1}\right) - \frac{3}{16h}\left(f_{i+2}-f_{i-2}\right) + \frac{1}{16}\left(s'(x_{i-2})+s'(x_{i+2})\right) \\ \iff s'(x_i) &= \frac{6}{7h}\left(f_{i+1}-f_{i-1}\right) - \frac{3}{14h}\left(f_{i+2}-f_{i-2}\right) + \frac{1}{14}\left(s'(x_{i-2})+s'(x_{i+2})\right) \end{aligned} \tag{3.20}$$

We notice that we have an equation similar to (3.20), so we can apply the same procedure as before: we use Equation (3.20) to approximate $s'(x_{i-2})$ and $s'(x_{i+2})$, and we use these approximations in (3.20).

(3.21)

$$\begin{aligned} s'(x_i) = \frac{6}{7h}\left(f_{i+1}-f_{i-1}\right) - \frac{3}{14h}\left(f_{i+2}-f_{i-2}\right) + \frac{1}{14}\Bigg(&\frac{6}{7h}(f_{i+3}-f_{i+1}+f_{i-1}-f_{i-3}) \\ &- \frac{3}{14h}\left(f_{i+4}-f_{i-4}\right) + \frac{1}{14}\left(s'(x_{i-4})+2s'(x_i)+s'(x_{i+4})\right)\Bigg) \\ \iff \left(1-\frac{2}{14^2}\right)s'(x_i) = \frac{78}{98h}(f_{i+1}-f_{i-1}) &- \frac{3}{14h}(f_{i+2}-f_{i-2}) + \frac{3}{49h}(f_{i+3}-f_{i-3}) \\ &- \frac{3}{14^2h}(f_{i+4}-f_{i-4}) + \frac{1}{14^2}(s'(x_{i+4})+s'(x_{i-4})). \end{aligned}$$

We will not detail the last step since it is the same procedure. We obtain the simplified version of (3.21)

$$s'(x_i) = \sum_{j=-8}^{8}\frac{\tilde{\omega}_j}{h}f_{i+j} + \frac{1}{14^2\alpha\beta}\left(s'(x_{i+8})+s'(x_{i-8})\right) \tag{3.22}$$

with $\beta = \left(\alpha - \frac{2}{14^2\alpha}\right)$ and $\alpha = 1-\frac{2}{14^2}$. We won't give the exact definition of the coefficients $\tilde{\omega}_j$, since they are long and unnecessary. However, they are constants independent of the function and the interpolation step. We will give an exact value to them later on. To complete the computation, we use the five points stencil finite differences to approximate the $s'(x_{i\pm8})$, which yield

$$\begin{cases} s'(x_{i+8}) &= \frac{1}{12h}\left(-f_{i+10}+8f_{i+9}-8f_{i+7}+f_{i+6}\right) \\ s'(x_{i-8}) &= \frac{1}{12h}\left(-f_{i-6}+8f_{i-7}-8f_{i-9}+f_{i-10}\right). \end{cases} \tag{3.23}$$

Using (3.23) in (3.22), we get the final cubic spline approximation of the Hermite slopes at the boundary $\{x_i\}$

$$s'(x_i) = \frac{1}{h}\sum_{j=1}^{10} \omega_j \left(f_{i+j} - f_{i-j}\right) \tag{3.24}$$

with the values of ω_j given in Table 3.1.

ω_1	8.03847585E-1	ω_6	-1.11379781E-3
ω_2	-2.15390339E-1	ω_7	3.01146127E-4
ω_3	5.77137695E-2	ω_8	-7.97151512E-5
ω_4	-1.54647393E-2	ω_9	1.77144780E-5
ω_5	4.14518786E-3	ω_{10}	-2.21430976E-6

Table 3.1: Value of the coefficients for the cubic-spline slopes approximation

Computation of the slopes in 2D

As we mentioned before, the procedure in 2D follows the same reasoning. However, as the computations are strenuous, we give here the first steps and the results. We set the number of discretization points the same on both directions. Let (x_i, y_i) be the point where we wish to compute the slopes. We start by the definition of the interpolated function $\tilde{f}(x_i, y_i)$ and its derivative along the x-axis

$$\begin{aligned} f_{i,i} = \tilde{f}(x_i, y_i) &= \sum_{j=-1}^{N+1}\sum_{k=-1}^{N+1} c_{j,k} B_j(x_i) B_k(y_i) \\ &= \frac{1}{36}\left(c_{i-1,i-1} + c_{i-1,i+1} + c_{i+1,i-1} + c_{i+1,i+1}\right) \\ &\quad + \frac{1}{9}\left(c_{i-1,i} + c_{i,i-1} + c_{i,i+1} + c_{i+1,i}\right) + \frac{4}{9}c_{i,i} \end{aligned} \tag{3.25a}$$

$$\begin{aligned} s_x(x_i, y_i) = \partial_x \tilde{f}(x_i, y_i) &= \sum_{j=-1}^{N+1}\sum_{k=-1}^{N+1} c_{j,k} B'_j(x_i) B_k(y_i) \\ &= \frac{1}{12h}\left(c_{i+1,i+1} + c_{i+1,i-1} - c_{i-1,i-1} - c_{i-1,i+1}\right) \\ &\quad - \frac{1}{3h}\left(c_{i-1,i} - c_{i+1,i}\right). \end{aligned} \tag{3.25b}$$

We proceed by injecting (3.25a) (which defines $c_{i,i}$) in (3.25b).

$$
\begin{aligned}
s_x(x_i, y_i) = \frac{1}{12h}\Bigg(&\frac{9}{4}f_{i+1,i+1} - \frac{1}{16}(c_{i,i} + c_{i,i+2} + c_{i+2,i} + c_{i+2,i+2}) \\
&- \frac{1}{4}(c_{i,i+1} + c_{i+1,i} + c_{i+1,i+2} + c_{i+2,i+1}) \\
&+ \frac{9}{4}f_{i+1,i-1} - \frac{1}{16}(c_{i,i-2} + c_{i,i} + c_{i+2,i-2} + c_{i+2,i}) \\
&- \frac{1}{4}(c_{i,i-1} + c_{i+1,i-2} + c_{i+1,i} + c_{i+2,i-1}) \\
&- \frac{9}{4}f_{i-1,i-1} + \frac{1}{16}(c_{i-2,i-2} + c_{i-2,i} + c_{i,i-2} + c_{i,i}) \\
&+ \frac{1}{4}(c_{i-2,i-1} + c_{i-1,i-2} + c_{i-1,i} + c_{i,i-1}) \\
&- \frac{9}{4}f_{i-1,i+1} + \frac{1}{16}(c_{i-2,i} + c_{i-2,i+2} + c_{i,i} + c_{i,i+2}) \\
&+ \frac{1}{4}(c_{i-2,i+1} + c_{i-1,i} + c_{i-1,i+2} + c_{i,i+1})\Bigg) - \frac{1}{3h}(c_{i-1,i} - c_{i+1,i}) \\
= \frac{3}{16h}&\left(f_{i+1,i+1} + f_{i+1,i-1} - f_{i-1,i-1} - f_{i-1,i+1}\right) \\
&- \frac{1}{16}\left(s_x(x_{i-1}, y_{i-1}) + s_x(x_{i-1}, y_{i+1}) + s_x(x_{i+1}, y_{i-1}) + s_x(x_{i+1}, y_{i+1})\right) \\
&- \frac{1}{48h}(c_{i+1,i} + c_{i+1,i+2} + c_{i+1,i-2} + c_{i+1,i} - c_{i-1,i-2} - c_{i-1,i} - c_{i-1,i} - c_{i-1,i+2}) \\
&- \frac{1}{3h}(c_{i-1,i} - c_{i+1,i}).
\end{aligned}
\tag{3.26}
$$

And we finish replacing the original terms of (3.25b)

$$
\begin{aligned}
s_x(x_i, y_i) = \frac{3}{16h}&\left(f_{i+1,i+1} + f_{i+1,i-1} - f_{i-1,i-1} - f_{i-1,i+1}\right) \\
&- \frac{1}{16}\left(s_x(x_{i-1}, y_{i-1}) + s_x(x_{i-1}, y_{i+1}) + s_x(x_{i+1}, y_{i-1}) + s_x(x_{i+1}, y_{i+1})\right) \\
&- \frac{1}{4 \times 12h}(c_{i+1,i} + c_{i+1,i+2} + c_{i+1,i-2} + c_{i+1,i} - c_{i-1,i-2} - c_{i-1,i} - c_{i-1,i} \\
&\quad - c_{i-1,i+2}) - \frac{1}{3h}\Bigg(\frac{9}{4}f_{i-1,i} - \frac{1}{16}(c_{i-2,i-1} + c_{i-2,i+1} + c_{i,i-1} + c_{i,i+1}) \\
&\quad - \frac{1}{4}(c_{i-2,i} + c_{i-1,i-1} + c_{i-1,i+1} + c_{i,i}) - \frac{9}{4}f_{i+1,i} \\
&\quad + \frac{1}{16}(c_{i,i-1} + c_{i,i+1} + c_{i+2,i-1} + c_{i+2,i+1}) \\
&\quad + \frac{1}{4}(c_{i,i} + c_{i+1,i-1} + c_{i+1,i+1} + c_{i+2,i})\Bigg)
\end{aligned}
\tag{3.27}
$$

$$
\begin{aligned}
s_x(x_i, y_i) = \frac{3}{16h}\left(f_{i+1,i+1} + f_{i+1,i-1} - f_{i-1,i-1} - f_{i-1,i+1}\right) - \frac{3}{4h}(f_{i-1,i} - f_{i+1,i}) \\
- \frac{1}{16}\left(s_x(x_{i-1}, y_{i-1}) + s_x(x_{i-1}, y_{i+1}) + s_x(x_{i+1}, y_{i-1}) + s_x(x_{i+1}, y_{i+1})\right) \\
- \frac{1}{4}(s_x(x_i, y_{i+1}) + s_x(x_i, y_{i-1}) + s_x(x_{i-1}, y_i) + s_x(x_{i+1}, y_i)).
\end{aligned}
\tag{3.28}
$$

After a few iterations, we get the final 2D cubic spline approximation of the Hermite slopes at the boundary $\{\mathbf{x}_{i,i}\}$, with the values of $\omega_{k,l}$ given in Table 3.2,

$$
s'(x_i, y_i) = \frac{1}{h}\sum_{k=1}^{4}\sum_{l=0}^{2}\omega_{kl}\left(f_{i-k,i-l} - f_{i+k,i+l}\right) - \omega_{kl}\left(f_{i+k,i-l} - f_{i-k,i+l}\right). \tag{3.29}
$$

k	$l=0$	$l=1$	$l=2$
1	-0.4113475177304965	-0.0141843971631206	0.00709219858156028
2	0.1152482269503545	0.0212765957446808	-0.00443262411347518
3	-0.02836879432624115	-0.0141843971631206	0.0
4	0.00398936170212766	0.00354609929078014	0.000443262411347518

Table 3.2: Value of the coefficients ω_{kl} for the 2D cubic-spline slopes approximation

Overview of SLMP the code

Before the development of the code, the environment of the implementation was largely discussed. On the one hand, GYSELA is an advanced and complex code, thus is not a suitable "play ground" for a Multi-patch code. On the other hand, SeLaLib was not prepared for such deep changes. Thus, we decided to start a code from scratch. As of today (September 2016), the code is over 3000 lines long and was developed in Python.

With the exception of the cubic spline interpolation code (which was written in SeLaLib in Fortran, and is connected to our code by an `f2py` interface), everything was created and tested separately and is adapted for the Multi-patch Approach. The general structure is sketched in Figure 3.9. Let us see briefly each module of the code.

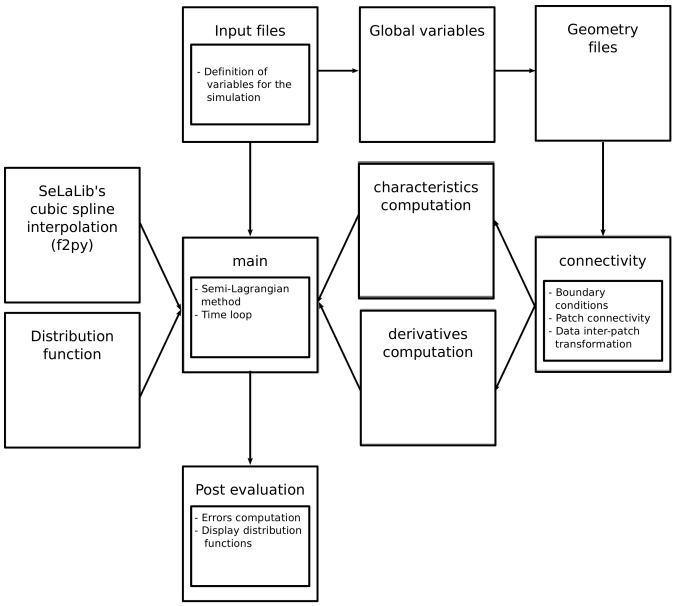


Figure 3.9: Semi-Lagrangian Multi-patch code structure

Input files: Every simulation takes a set of parameters which are defined in this module. There are different input files for every simulation.[4] The variables to define are: the domain, the discretization in space and time, the model to study and the associated parameters (value of advection, ...), the initial distribution function, the type of interpolation to use, etc.

Distribution function: Multiple distribution functions are defined in this file. They also adapt to the domain studied.

SeLaLib's cubic spline interpolation: Needs a compiled library of SeLaLib. Interface between SeLaLib's code in Fortran and our python code. Uses `f2py`.

Global variables: All variables for a simulation have a default value that can be found in this module, as well as some complementary global variables needed throughout the simulation.

Geometry files: The geometry files are not only the output files created by CAID (in `.xml` format), but also the files that allow the reading, loading and translations for our code. The structures for the geometry, and the transformations are created here.

[4] In fact, every test-case presented below, has its own input files to facilitate the reproduction of the results found in this manuscript.

Connectivity: Defines the boundary condition that CAID cannot define. Contains some of the most important functions such as: `get_neighbours` (which returns all the neighbors of a given patch at each face), `get_face` (which returns the data of a patch at a given face), `transform_advection` (which transforms an advection from one patch to another one).

Characteristics computation: A characteristic, in our code, is a set of coordinates associated to a given patch. The script in this part computes the inter-patch back-tracing of the characteristics.

Derivatives computation: Computes the slopes at the boundaries of each patch of the domain using the methods described in the last section.

First Multi-patch results

Before even testing the method in a circular domain, we test it in a simple Multi-patch domain. Studying the convergence of the error will show if the order of the methods is conserved when binding the patches. The simplest domain that we can think of is a 2 patch decomposition, similar to Figure 3.7. The first coordinate transformation, F^0, is the identity, and F^1 is a translation of 1 along the x-axis.

The constant advection equation

Our first model will consist of a constant advection equation (3.2).

Test-case 1: Constant advection of sinusoidal distribution on a 2MP rectangular domain

For a first test, we consider the following initial distribution

$$f_0(x, y) = \cos(2\pi x)\sin(2\pi y)$$

which representation can be seen in Figure 3.10 (left). This test-case has the property will be a perfect first test since it is a periodic function and the proper definition of the internal and external boundary conditions are important to get a correct approximation.

For the preliminary tests, we discretize with the same number of points both in x and y directions, and for both patches. As parameters for the computation we choose the time step $\Delta t = 0.05$, and an advection along the x axis of 0.1, and we implemented periodic boundary conditions between the two external faces in x, yielding the connectivity lists

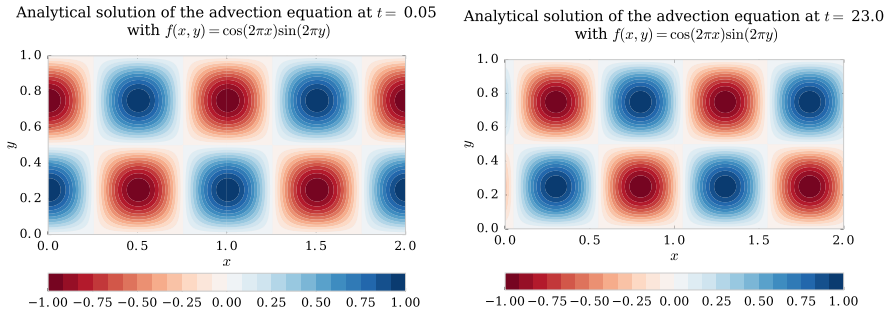
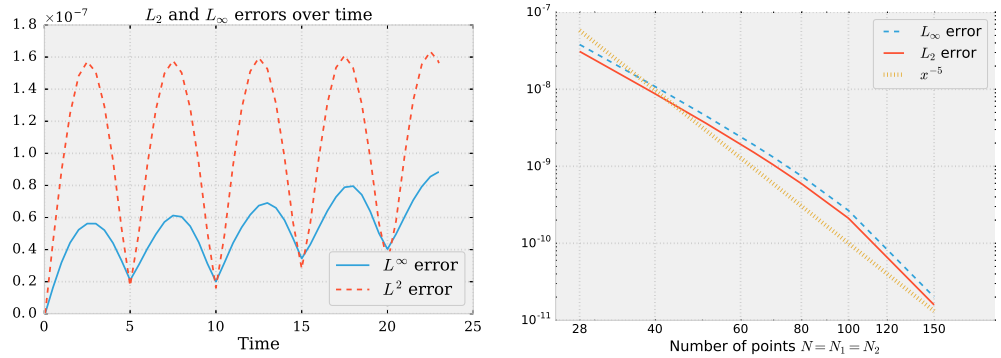


Figure 3.10: Test-case 1: distribution function at initial and final time

```
list_conn1 = [ [0,1], [0,3] ] list_conn2 = [ [1,3], [1,1] ].
```

The final time for the simulation is $t_{max} = 23$, thus 460 time steps are computed. The approximated distribution function computed using the Multi-patch Semi-Lagrangian scheme is in Figure 3.10 (right).

No oscillations or numerical artifacts visible in the computed solution, and the limit between the two patches (at $x = 1$) is invisible when plotting the distribution function. Therefore, we need to study the errors of the simulation, to see the convergence of the method, as well as, the properties conserved.

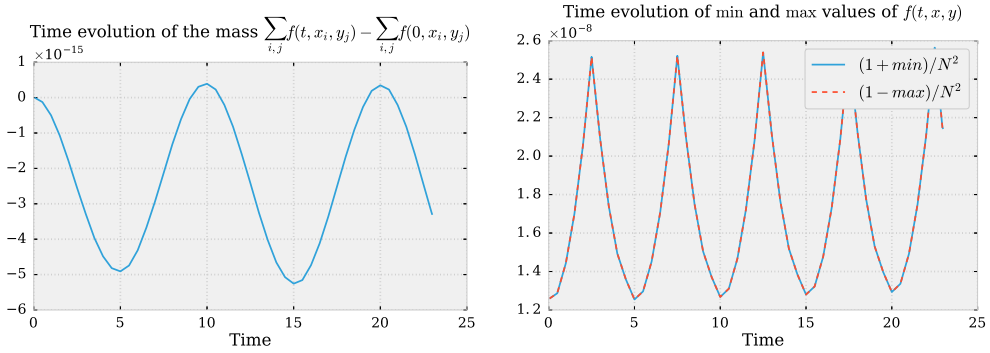


(a) Errors computed with $N = 30$ **(b)** Errors computed at time $t_{max} = 23$

Figure 3.11: Test-case 1: Evolution of L_2 and L_∞ errors over time (left) and number of cells (right)

In Figure 3.11 we plotted the errors with L_2 norm, computed using the formula $\Delta x^2 \left(\sum_i |f_{num}(t, \mathbf{x}_i) - f(t, \mathbf{x}_i)|\right)^{1/2}$ with f_{num} the computed solution and f the exact solution, and the L_∞ error is computed $\Delta x^2 \left(\max_i |f_{num}(t, \mathbf{x}_i) - f(t, \mathbf{x}_i)|\right)$. We see that both errors grow with time, which is expected when using the Semi-Lagrangian method. Furthermore, in Figure 3.11b, we see that the method con-

verges for both errors with order 5, which is the order of the method used for computing the origin of the characteristic and the slopes.



(a) Mass evolution over time

(b) Minimal value evolution

Figure 3.12: Test-case 1: Time evolution of the mass and the minimal value for a simulation using $N = 40$ and $\Delta t = 0.05$

While the mass of the distribution function is correctly conserved (error of order 10^{-14} with only $N = 40$ cells per direction, see Figure 3.12a), the minimum value conservation is less precise (see Figure 3.12b). Due to the particularity of the distribution function and the model studied, we conclude that these preliminary results are satisfactory and we can proceed to other test-cases and models.

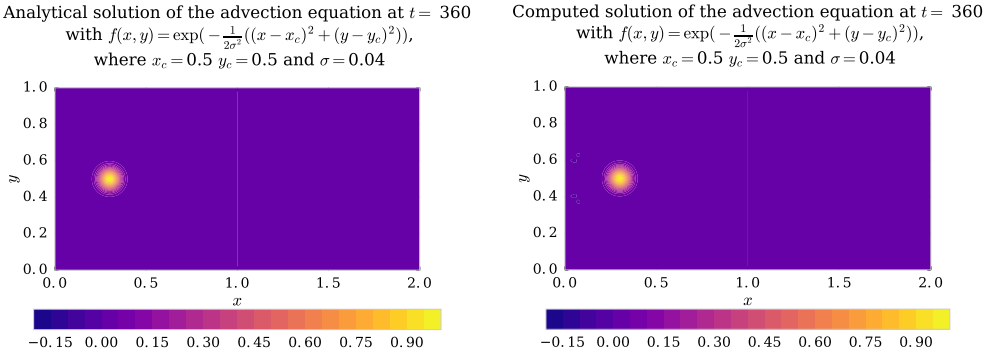
Test-case 2: Constant advection of Gaussian pulse on a 2MP rectangle

Our second test-case, for the same constant advection model along the x-axis and with the same domain discretization, is a Gaussian pulse, which initial distribution is

$$f_0(x,y) = \exp\left(-\frac{1}{2}\left(\frac{(x-0.5)^2}{0.04^2} + \frac{(y-0.5)^2}{0.04^2}\right)\right). \tag{3.30}$$

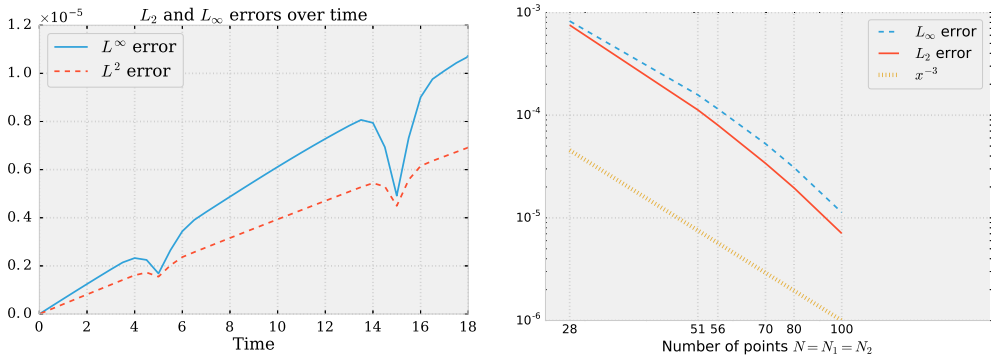
This function has a great gradient around the pulse, which makes it an interesting test-case for the internal boundary conditions. We choose a simulation of 360 iterations, when $t_{max} = 18$ and the pulse–initially on patch $\mathcal{P}_0$–has gone through the second patch $\mathcal{P}_1$ and is back on $\mathcal{P}_0$.

We notice that, in Figure 3.13b, some oscillations have appeared. In fact, the computed solution has negative values that are not in Figure 3.13a. These oscillations appeared already at the first iteration which means that they are generated by the interpolation method. Thus, these artifacts are not generated by the Multi-patch Approach. We also noticed, the oscillations are not amplified when crossing patches.



(a) Analytic distribution at $t_{max} = 18$ **(b)** Approximative distribution at $t_{max} = 18$

Figure 3.13: Test-case 2: Distribution function at time $t_{max} = 18$ with $N = 100$

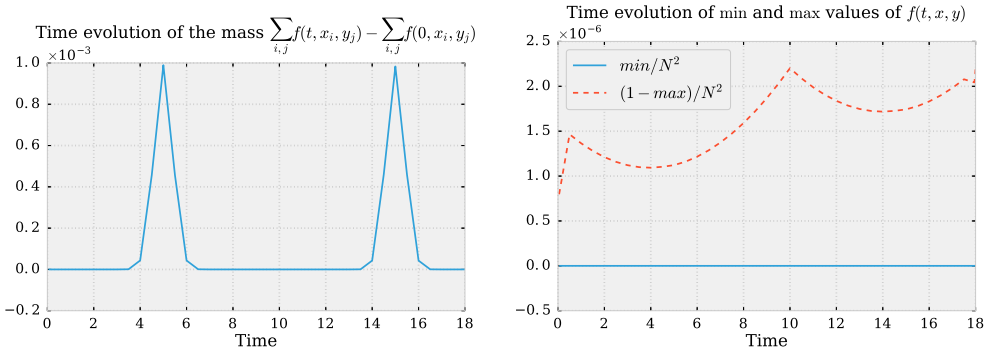


(a) Errors computed with $N = 100$ **(b)** Errors computed at time $t_{max} = 18$

Figure 3.14: Test-case 2: Evolution of L_2 and L_∞ errors over time (left) and number of cells (right)

Further proof that the Multi-patch Approach did not amplify the errors is Figure 3.14a. The critical moments of the simulations are around $t = 0.5$ and $t = 1.5$, which represent the time when the Gaussian pulse moved from one patch to another. It is impossible, with the chosen resolution, to identify these moments when observing the L_∞ error. On the other hand, the L_2 error shows some low points at those times. As the cubic-spline interpolation has the property of dissipating the solution over time, we suppose these low peaks are due to the 5-th order approximation of the slopes at the boundaries. Thus, we get solutions that are more accurate at those critical points. In Figure 3.14, we see that, since the interpolation method is not exact for this test-case, we found its order (3) as the order of our scheme.

The transition between patches can also be seen when studying the minimal value of the distribution function (see Figure 3.15b). Nevertheless, the conserva-



(a) Mass evolution over time

(b) Minimal value evolution

Figure 3.15: Test-case 2: Time evolution of the mass and the minimal value for a simulation using $N = 100$ and $\Delta t = 0.05$

tion of positivity is satisfactory for such a test-case. The mass conservation (see Figure 3.15a) seems more troubled with peaks going to 10^{-3}. However, shortly after the pulse went through the boundaries, the mass conservation is as good as a single patch simulation. This is probably due to the slopes approximation, which basically act as fluxed when the distribution is non-null near the boundaries, but averages out through time.

Test-case 3: Constant advection of Gaussian pulse on a 4MP square domain

For the third test-case, we change the domain configuration. We keep a squared domain, where the transformation functions from the logical to the physical domain are the identity function plus a translation. However, we increase the number of patches to four and we organized them so that there is a point where the four patches meet. A sketch of the configuration can be seen in Figure 3.16. Furthermore, we set the advection vector to $\mathbf{A} = (0.1, 0.1)^T$.

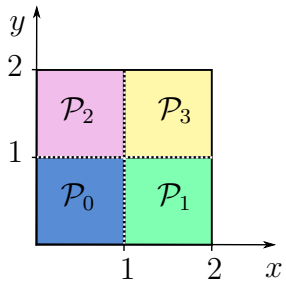
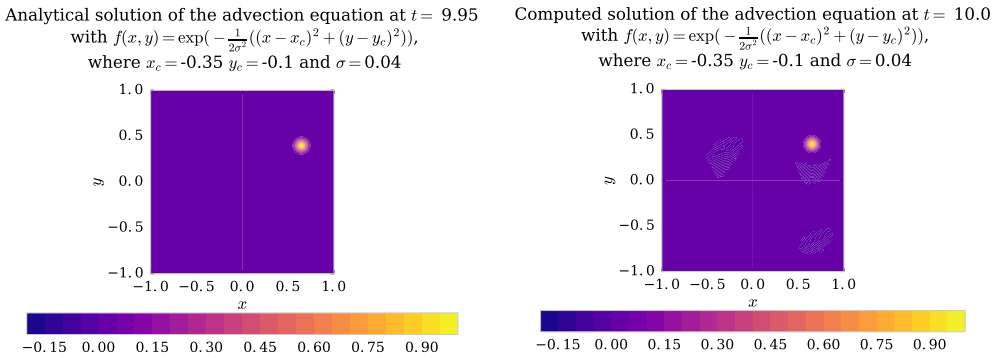


Figure 3.16: Square domain discretized in four identical patches

The same distribution function as in Test-case 2, a Gaussian pulse, is used. In Figure 3.17, we can see that as in Test-case 2, the numerical approximation presents some waves around the pulse even with such a fine discretization.



(a) Analytic distribution at $t_{max} = 10$ **(b)** Approximative distribution at $t_{max} = 10$

Figure 3.17: Test-case 3: Distribution function at time $t = 10$ with $N = 100$

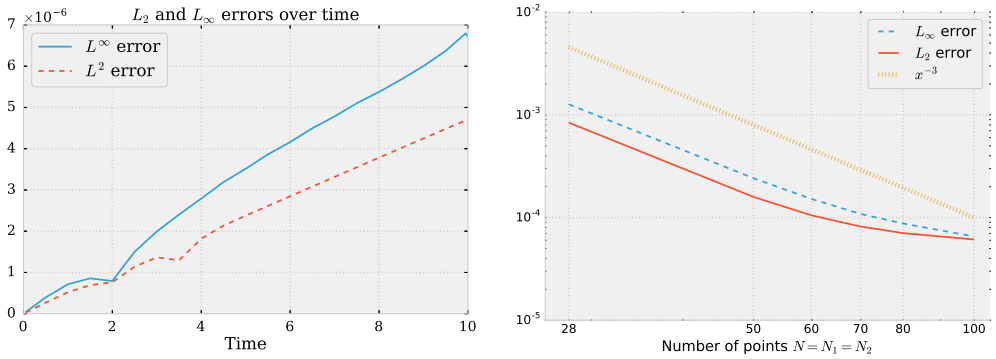
The errors plotted in Figure 3.18a show the exact time of the critical moment when the bulb goes through the center point of the domain. However, the errors drop again after the transition time is over. This is due to the fact that the slope computation is not exactly symmetrical, at the critical time. Whereas, when the pulse is away from the boundaries the slopes are again exact (and equal to 0) and the dissipation error takes over.

We can see the same phenomenon in the mass evolution. More interestingly Figure 3.18b, shows that for simulations of 60 points or less, the scheme is of order 3 as expected. Even if for finer discretizations there is stagnation, this is probably due to the approximation done at the corners of the patches.

For this test-case, we also plotted the mass conservation, Figure 3.19a, and the minimal value of the distribution function, Figure 3.19b. While the first one has a behavior similar to previous test-cases, the second one is almost linearly decreasing. However, the order of magnitude of the error for the minimal value is the same or better than the previous ones. Since the positivity is not conserved, this impacts hardly on the mass conservation. A further study on the slopes approximation should be done in order to conserve the mass exactly.

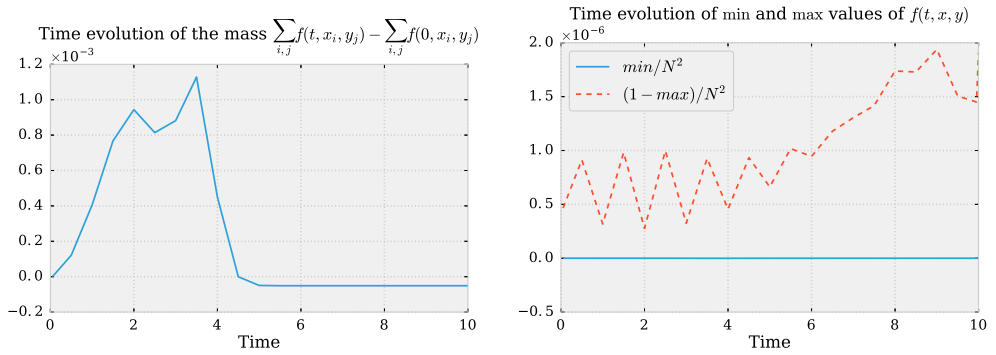
Test-case 4: Constant advection of pulse through singular points of 5MP disk

For this Test-case, we use the same geometry, defined in Chapter 3, as well as the same model. However, the Gaussian pulse will go through two of the singular points of the mesh. The objective of this Test-case is to see how the treating



(a) Errors computed with $N = 100$ **(b)** Errors computed at time $t_{max} = 10$

Figure 3.18: Test-case 3: Evolution of L_2 and L_∞ errors over time (left) and number of cells (right)



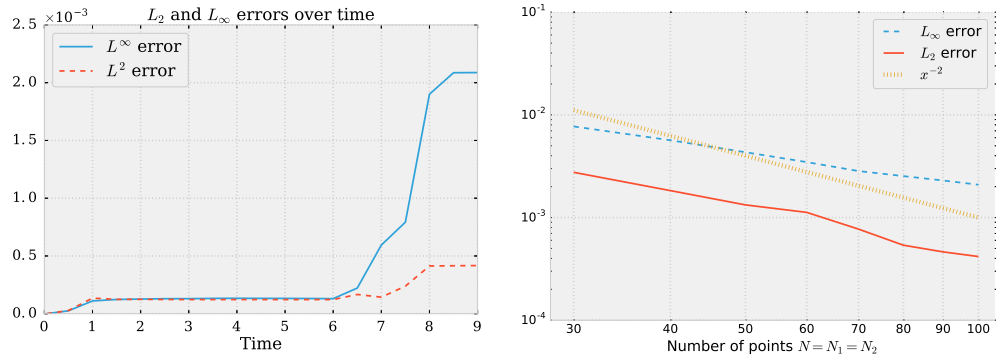
(a) Mass evolution over time **(b)** Minimal and maximal values evolution

Figure 3.19: Test-case 3: Time evolution of the mass and the bound values for a simulation using $N = 100$ and $\Delta t = 0.004$

of the null jacobian affects the results of the simulation. The parameters of the simulation are $t_{max} = 9$, $\Delta t = 0.05$, $\mathbf{x}_c = (-0.65, 0)$, $\mathbf{A} = (0.15, 0)$.

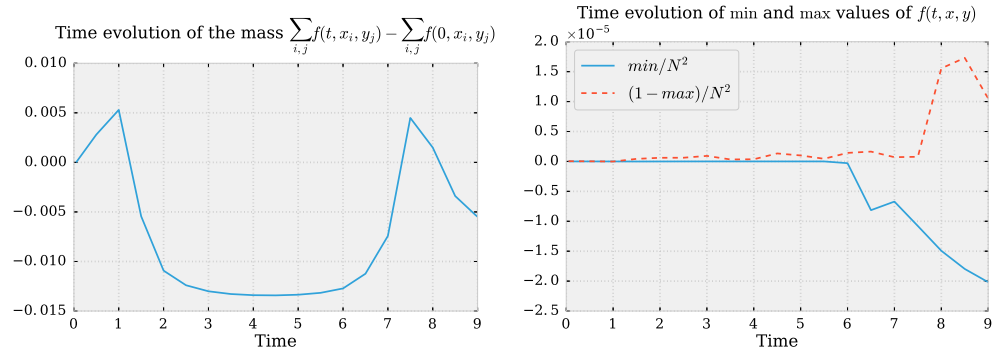
In Figure 3.20a, we can see that there are 2 times when the error norms have a significant increase. These two moments correspond to the moments when the Gaussian pulse goes through the singular points. We also notice in Figure 3.20b, the convergence error is completely lost, and we only have order 2 convergence.

The mass evolution also shows two peaks generated by the singularity. As for the minimal and maximal value evolution over time, we cannot really conclude anything. We should mention that the stress of a pulse going through this singular point is not a common physical situation in plasmas. However, we hope that the solution we provide should approximate the solution better.



(a) Errors computed with $N = 100$ **(b)** Errors computed at time $t_{max} = 10$

Figure 3.20: Test-case 4: Evolution of L_2 and L_∞ errors over time (left) and number of cells (right)



(a) Mass evolution over time **(b)** Minimal and maximal values evolution

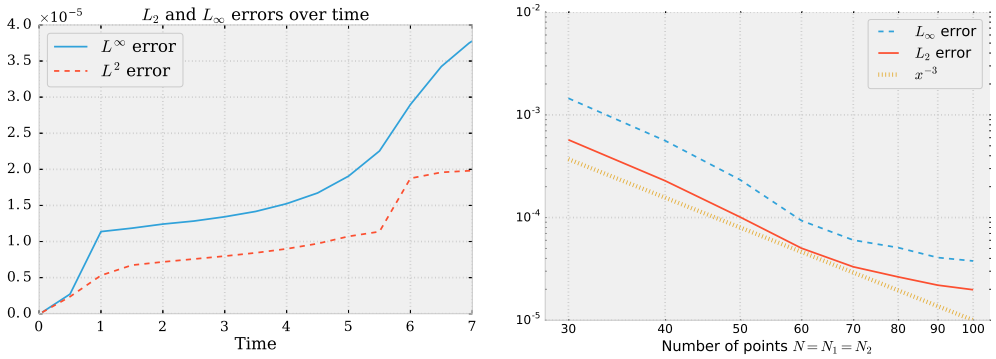
Figure 3.21: Test-case 4: Time evolution of the mass and the bound values for a simulation using $N = 100$ and $\Delta t = 0.05$

Test-case 5: Constant advection of pulse through no singular points of 5MP disk

To verify the impact of the singular points on the simulation we need to compare it to an advection that does not go through any singular point.

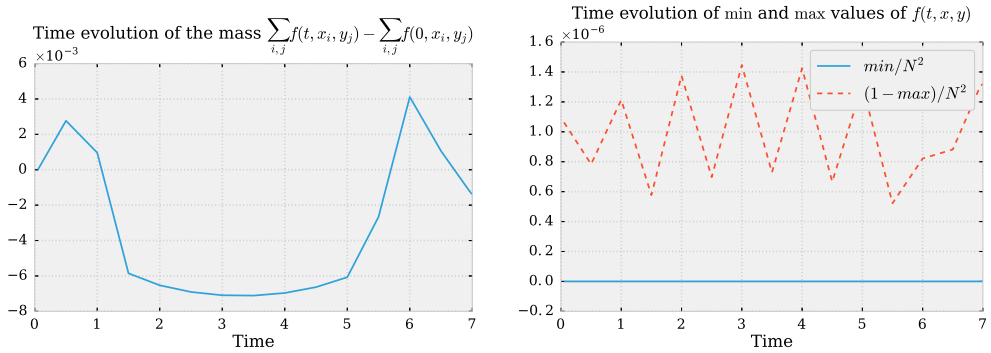
Thus, for this Test-case, we choose a slightly shifted initial distribution and a diagonal advection to make sure the advection is not along any of the meshes directions. The parameters of this simulation are: $t_{max} = 7$, $\Delta t = 0.05$, $\mathbf{x}_c = (-0.5, -0.5)$, $\mathbf{A} = (0.15, 0.15)$.

We can see that even if the mass and error norms present some bumps when the pulse goes through the patches, the discrepancies are minor and the errors are slower. More importantly, the convergence of the simulation respects the order of the interpolation, thus the multiple-patch approach does not affect the global



(a) Errors computed with $N = 100$ **(b)** Errors computed at time $t_{max} = 7$

Figure 3.22: Test-case 5: Evolution of L_2 and L_∞ errors over time (left) and number of cells (right)



(a) Mass evolution over time **(b)** Minimal and maximal values evolution

Figure 3.23: Test-case 5: Time evolution of the mass and the bound values for a simulation using $N = 100$ and $\Delta t = 0.05$

simulation. This means that the only real problem is around the singular points. Even if these results are not promising, we decide to study more complex models.

Solving the variable coefficient advection equation with circular motion

We wish now to have a circular motion advection. Let us remember the general advection equation:

$$\partial_t f(x, y) + \nabla \cdot (\mathbf{A} f) = 0, \tag{3.31}$$

with initial condition

$$f(0, x, y) = f_0(x, y).$$

The advection vector $\mathbf{A}$ defined as $\mathbf{A} = \begin{pmatrix} -2\pi y \\ 2\pi x \end{pmatrix}$ will define a circular advection in a counter-clockwise motion. We can notice that $\mathbf{A}$ is divergence free, and we know that $\nabla \cdot (\mathbf{A}f) = \mathbf{A} \cdot \nabla f + f \, \nabla \cdot \mathbf{A}$, so we obtain

$$\frac{\partial f}{\partial t}(x, y) + \mathbf{A} \cdot \nabla_{\mathbf{x}} f(x, y) = 0. \tag{3.32}$$

Solving the equation on the physical space

We will use the method of characteristics to solve the model, as we have done previously. It yields the following system.

$$\begin{cases} \dfrac{\partial X_1}{\partial t} &= -2\pi y \\ \dfrac{\partial X_2}{\partial t} &= 2\pi x \end{cases} \quad \text{with the initial condition} \quad \begin{cases} X_1(s) &= x^0 \\ X_2(s) &= y^0 \end{cases}.$$

A solution is given by

$$\begin{cases} X_1(t) &= r_0 \cos(2\pi t + \theta_0) \\ X_2(t) &= r_0 \sin(2\pi t + \theta_0) \end{cases} \tag{3.33}$$

where r_0 and θ_0 are the radius and angle of the initial solution (x^0, y^0).

Solving the equation on the patch

Following the same procedure as in the latter section, we introduce another notation for the advection coefficient. Introducing the function $\Psi = -\pi x^2 - \pi y^2$, we can notice that (3.31) can be written as follows.

$$\frac{\partial f}{\partial t}(x, y) + \overrightarrow{rot} \, \Psi \cdot \nabla_{\mathbf{x}} f(x, y) = 0. \tag{3.34}$$

To transform the equation to general coordinates, we can follow the same steps as before, and we obtain

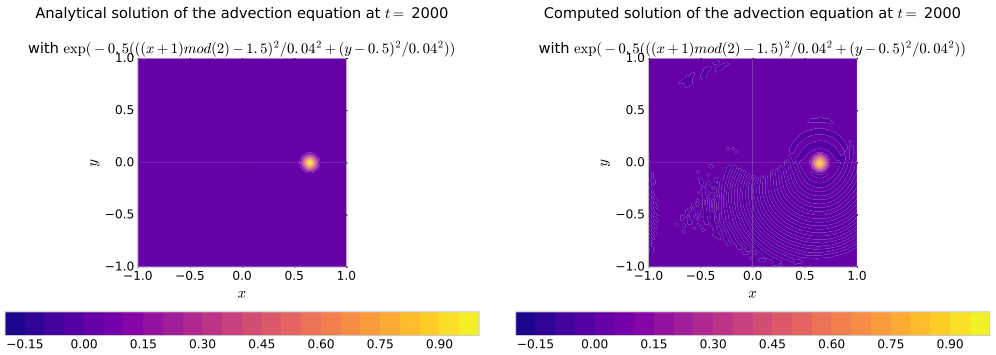
$$\frac{\partial \tilde{f}}{\partial t}(\eta, \xi) + \frac{1}{|\mathbf{J}|} \overrightarrow{rot}_{\boldsymbol{\eta}} \, \tilde{\Psi} \cdot \nabla_{\boldsymbol{\eta}} \tilde{f}(\eta, \xi) = 0. \tag{3.35}$$

We can notice it has the same structure as equation (3.13). Therefore, the method for solving it will be the same. As the advection is circular, this test-case will be particularly interesting to test over long time.

Numerical Results

Test-case 6: Circular advection of Gaussian pulse on a 4MP square domain

For the next test-case, we study the model defined in (3.32), with initial condition a Gaussian pulse Equation (3.30) with the following parameters: $\sigma = 0.04$, and $\mathbf{x}_c = (0.65, 0)$. We will keep the same geometry as in Test-case 3 (*i.e.* a square domain decomposed in four squared patches). We use a smaller time step than in the previous test-cases, $\Delta t = 0.0005$, since the advection coefficient is already multiplied by 2π. The final $t_{max} = 1$ has been chosen as it corresponds to the time for the pulse to go through the four inter-patch boundaries.

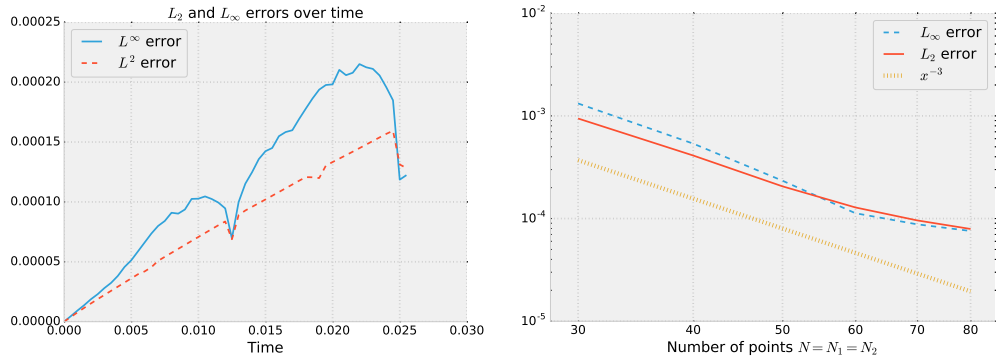


(a) Analytic distribution at $t_{max} = 1$ **(b)** Approximative distribution at $t_{max} = 1$

Figure 3.24: Test-case 6: Distribution function at time $t = 1$ with $N = 60$

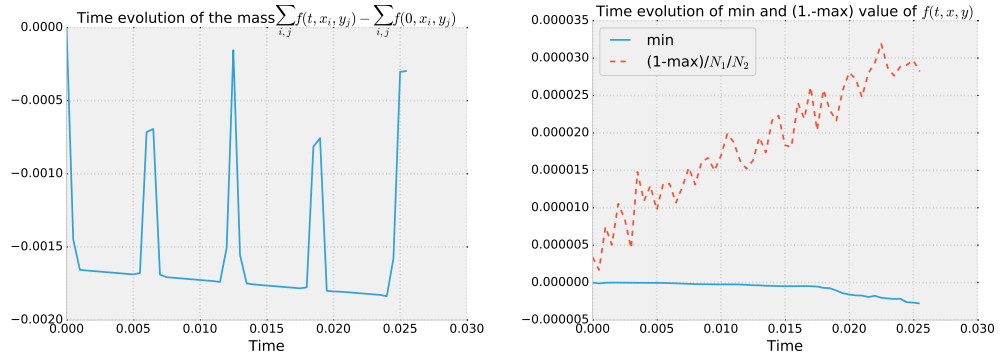
We see that even if the two solutions are close in Figure 3.24, the oscillations created by the interpolation scatter along the domain. Since this domain is periodic both in the vertical and horizontal directions, they scatter quickly. We suppose that with a higher degree interpolation method, or with a distribution function with a more constant gradient, we would not see this phenomenon.

On the one hand, for the errors plotted in Figure 3.25a, there is not much we can conclude other that the error seems to increase linearly with respect to time. However, this behavior is similar to what was observed in Test-case 2. In Figure 3.25b, we see that the method is of order 3, which is what we were expecting. On the other hand, the mass and extremum values, in Figure 3.26, seem to behave normally.



(a) Errors computed with $N = 60$ **(b)** Errors computed at time $t_{max} = 1$

Figure 3.25: Test-case 6: Evolution of L_2 and L_∞ errors over time (left) and number of cells (right)



(a) Mass evolution over time **(b)** Minimal value evolution

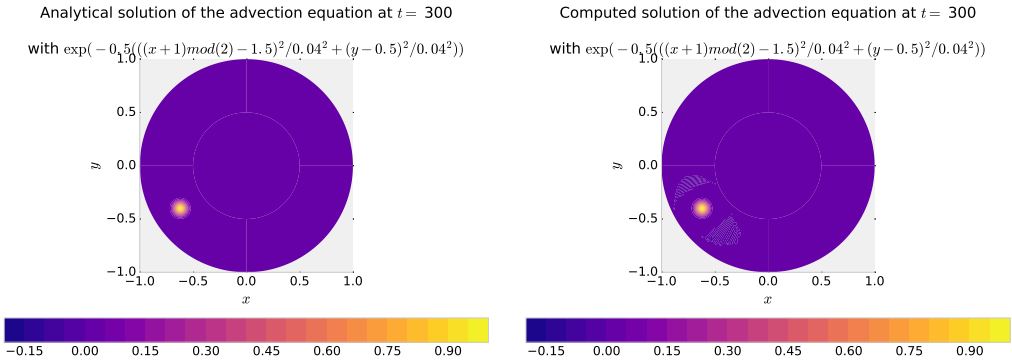
Figure 3.26: Test-case 6: Time evolution of the mass and the minimal value for a simulation using $N = 60$ and $\Delta t = 0.0005$

Test-case 7: Circular advection of Gaussian pulse on 5MP disk domain

We now use a circular domain, as defined on Chapter 3, decomposed on five patches. The distribution function is the same as in the last domain – a Gaussian pulse of width $\sigma = 0.04$ and centered at $(x_c, y_c) = (0.62, 0.4)$ (a point on the external crown) – which will be displaced with a circular advection with $\Delta t = 0.005$. By the time $t_{max} = 1.5$, the Gaussian passes 6 patch interfaces. For this test case we show the final distribution to show that the difference between the two results is not noticeable by the naked eye.

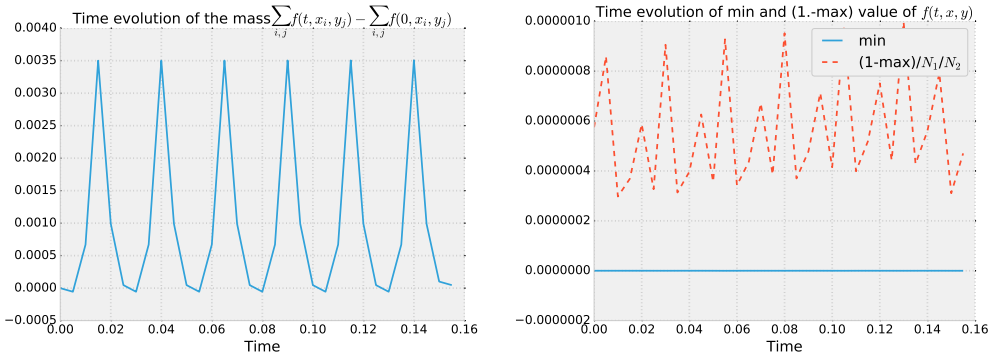
The evolution of the mass (Figure 3.28a) and of the errors with respect to the time (Figure 3.29a), show that the passage between patches is not completely smooth.

In Figure 3.28b, we plotted not only the evolution of the minimal value, but



(a) Analytic distribution at $t_{max} = 1.5$ **(b)** Computed distribution at $t_{max} = 1.5$

Figure 3.27: Test-case 7: Distribution function at time $t = 1.5$ with $N = 100$



(a) Mass evolution over time **(b)** Minimal value evolution

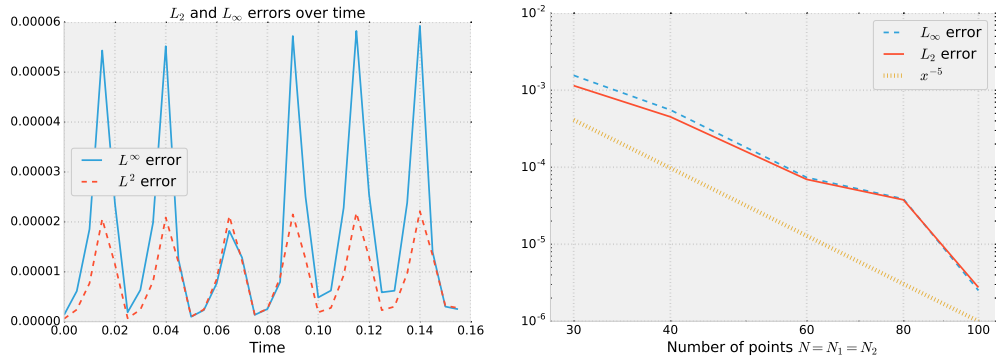
Figure 3.28: Test-case 7: Time evolution of the mass and the min and max values, using $N = 100$ and $\Delta t = 0.005$

also the maximal value, which for this test-case is less accurate than the minimal value. Both errors are minimal.

Finally, in Figure 3.29b, we see that the whole simulation is of order 5. Nevertheless, since here the distribution was null around the singular points along the whole simulations, this is not surprising.

Test-case 8: Circular advection of Gaussian pulse through singular points on 5MP disk domain

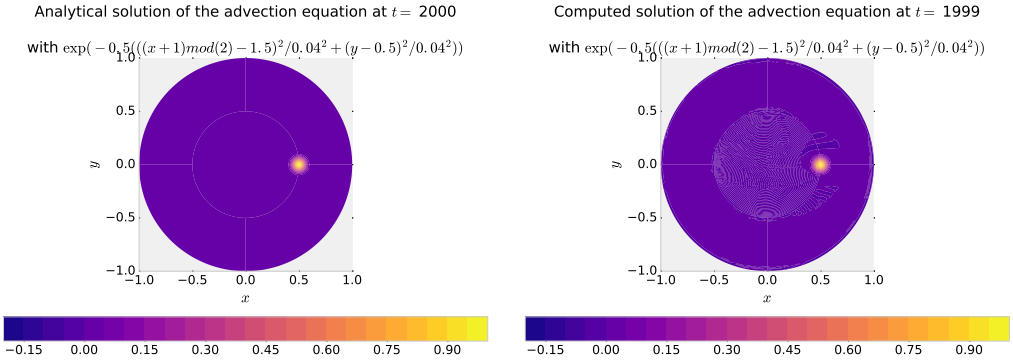
From Test-cases 4 and 5, we saw the difference between the simulations where the distribution goes through the singular points and when it does not. Let us see with this test if the singular points still play an important role. The simulation is also based on a circular advection with $\Delta t = 0.0005$. The initial distribution is also a



(a) Errors computed with $N = 100$ **(b)** Errors computed at time $t_{max} = 1.5$

Figure 3.29: Test-case 7: Evolution of L_2 and L_∞ errors over time (left) and number of cells (right)

Gaussian pulse with $\sigma = 0.04$, $\mathbf{x}_c = (0.5, 0)$ (which is exactly a singular point of the mesh). We let the simulation run until $t_{max} = 1.$, the equivalent of 4 patches interfaces to be crossed, thus the pulse goes through all the singular points.

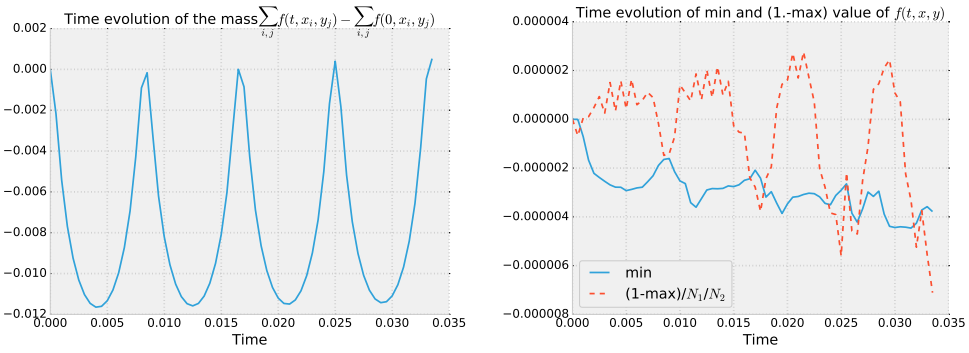


(a) Analytic distribution at $t_{max} = 1.5$ **(b)** Computed distribution at $t_{max} = 1.5$

Figure 3.30: Test-case 8: Distribution function at time $t = 1.5$ with $N = 80$

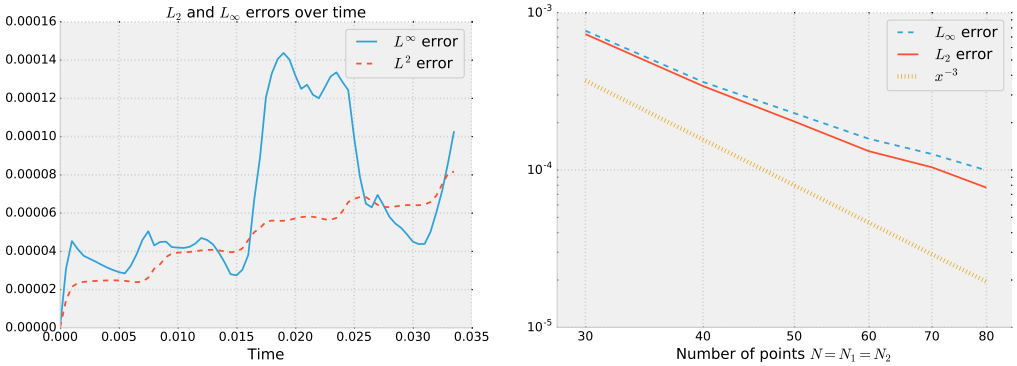
On the one hand, Figure 3.31a shows some peaks around the moments of crossing of the singular points. Incidentally, these peaks bring the mass conservation closer to the exact value. However, this is a random phenomenon. The minimum and maximum values, on the other hand, do not show any clear impact at the critical moments.

The evolution of the L_2 and the L_∞ errors do not present any visible consequence of the singular points. In fact, the L_2 error increases apparently linearly. The convergence rate of the global simulation went back down to 3, which is still the minimal order we expect from our scheme.



(a) Mass evolution over time **(b)** Minimal value evolution

Figure 3.31: Test-case 8: Time evolution of the mass and the min and max values, using $N = 80$ and $\Delta t = 0.005$



(a) Errors computed with $N = 80$ **(b)** Errors computed at time $t_{max} = 1.5$

Figure 3.32: Test-case 8: Evolution of L_2 and L_∞ errors over time (left) and number of cells (right)

Even if the singular points are not obviously affecting the results, we decide to explore further solutions.

Alternative meshes without a singular point

As we have seen on (3), the previously described mesh has four singular points. And this leads to some instabilities in the singular points' neighborhood. Therefore, we try to define other meshes without any singular points. However, all the solutions presented in this section have one common point: an external polar mesh. Since this is the current discretization in GYSELA, the solution we choose needs to present a polar crown.

First alternative: a shortcut?

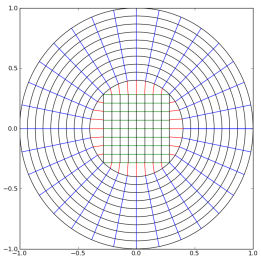


Figure 3.33: First alternative mesh: the crowned square

The singular points shown before come from the transformation of a square to a circle. Having those singularities is inevitable in such a mapping. Therefore, our first idea is not to use any transformation at all on the internal mesh. Thus, we keep it as a square, see Figure 3.33.

We realized that while using the former mapping, the data on the circle which separates the polar annulus of the interior mesh, was saved and computed twice, once for the external patches and one for the internal one. As this data is redundant, we should be able to skip it and have an artificial buffer zone, where there is no actual mesh. The problem with this first attempt was that the back-tracing of the particles was too costly since we broke the "edge-to-edge" rule, explained in Chapter 3. The second, and fundamentally negative aspect of this mesh, is that the buffer zone, has its own transformation function, which is unknown and it is not taken into account when considering only two mappings. This yielded results distorted when the distribution function was not null on those zones.

We concluded that, unless some special development was made for the "ghost" cells, the domain discretization needed to be continuous all along Ω.

Second alternative: tweaking the 5-patch configuration

One of the benefits of using the IGA approach, and a tool such as CAID, is the ability of tweak and modify the meshes as precisely as necessary to obtain the desired discretization. From this, we decided to test a mesh similar to our original 5-patch disk, but where the interior disk is "pinched" at the four singular points, stretching it out. This way, the Jacobian on these points is not null. The mesh resulting from this modification is on Figure 3.34.

The resulting mesh is thus continuous, with no singular points, and conserves a perfect polar mesh in the external crown. We could consider four additional patches as an extra exterior crown, that will be mapped with the polar mesh and that are definitely not modified by the pinching.

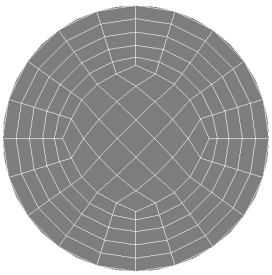


Figure 3.34: Second alternative mesh: the pinched disk

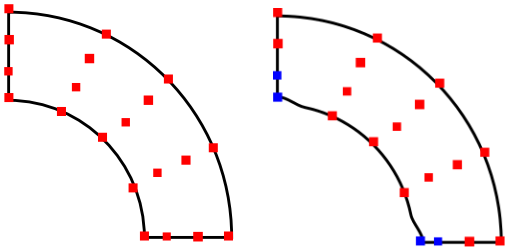


Figure 3.35: Sketch moving quart-annulus control points

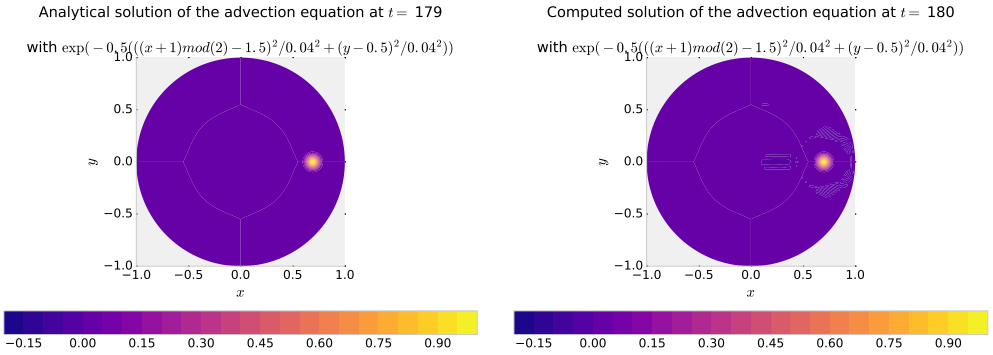
Since this geometry is not common, we had to create it. The code was implemented in CAID and now it is accessible from the user interface. Let us explain the procedure quickly: We start by creating a quarter annulus, of degree 3 and with 4 control points in each direction (see Figure 3.35). Then we modify two points of the internal corners of the quart-annulus. Let ε be the coefficient of stretching. We apply this procedure to the four external patches and analogously to the internal patch. The default value of stretching is $\varepsilon = 0.5$.

Test-case 9: Constant advection of Gaussian pulse on the 5MP pinched disk

Firstly, we apply on this configuration (with the default stretching coefficient $\varepsilon = 0.5$) a constant advection that goes through two of the critical points. Similar to what we did in Test-case 4. Let $t_{max} = 9$, $\Delta t = 0.05$, $\mathbf{x} = (-0.65, 0)$ and $\mathbf{A} = (0.15, 0)$.

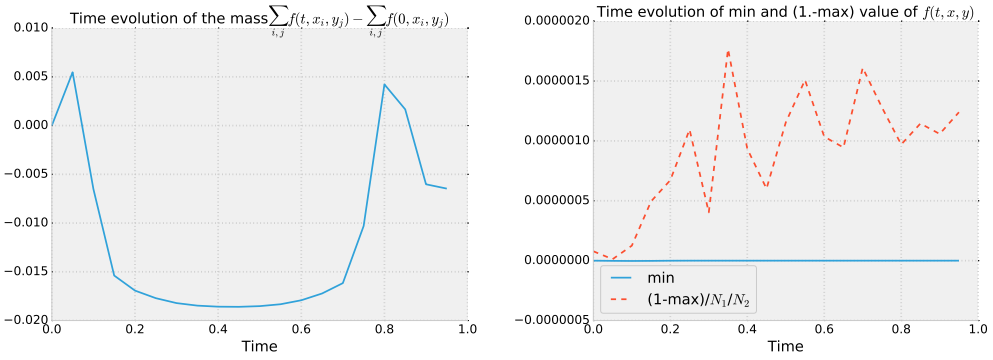
The distribution function still presents some oscillations around the singular points in Figures 3.36a and 3.36b. However, the actual bulb is really close to the analytical solution.

The minimal and maximal value evolution (Figure 3.37b) are much better as the one on the classical (*i.e.* un-pinched) 5MP disk domain. This is due to the fact that the singularity causes oscillations and thus maximal and minimal value are not conserved as good as they are in this configuration. The mass conservation does not present a significant difference between the two domains discretization.



(a) Analytic distribution at $t_{max} = 9$ **(b)** Computed distribution at $t_{max} = 9$

Figure 3.36: Test-case 9: Distribution function at time $t = 9$ with $N = 100$



(a) Mass evolution over time **(b)** Minimal value evolution

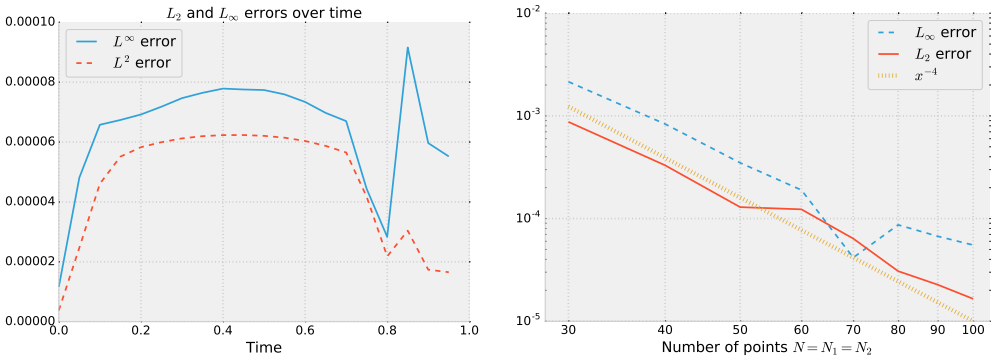
Figure 3.37: Test-case 9: Time evolution of the mass and the minimal and maximal values, using $N = 100$ and $\Delta t = 9$

More importantly, the improvement is shown in Figure 3.38. Compared to the classical 5MP disk, we gained an order of the convergence. And the evolution of the error norms with respect to time, is also more precise on this configuration. We proceed to more complex configurations.

Test-case 10: Circular advection of Gaussian pulse on the 5MP pinched disk

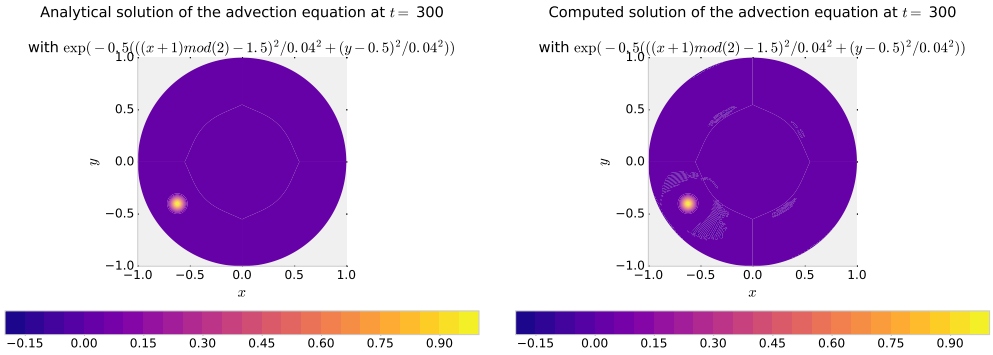
This test-case is analogous to Test-case 7, we advect a Gaussian pulse through the external patches. We set $t_{max} = 1.5$ and $\Delta t = 0.005$. As for the initial distribution, let $\sigma = 0.04$ and $\mathbf{x}_c = (0.62, 0.4)$. We chose to use the same stretching coefficient $\varepsilon = 0.5$.

We notice that the difference between these results and the one of Test-case 7 are minimum. In fact the difference between the mass, the minimum and the



(a) Errors computed with $N = 100$ **(b)** Errors computed at time $t_{max} = 1.5$

Figure 3.38: Test-case 9: Evolution of L_2 and L_∞ errors over time (left) and number of cells (right)



(a) Analytic distribution at $t_{max} = 1.5$ **(b)** Computed distribution at $t_{max} = 1.5$

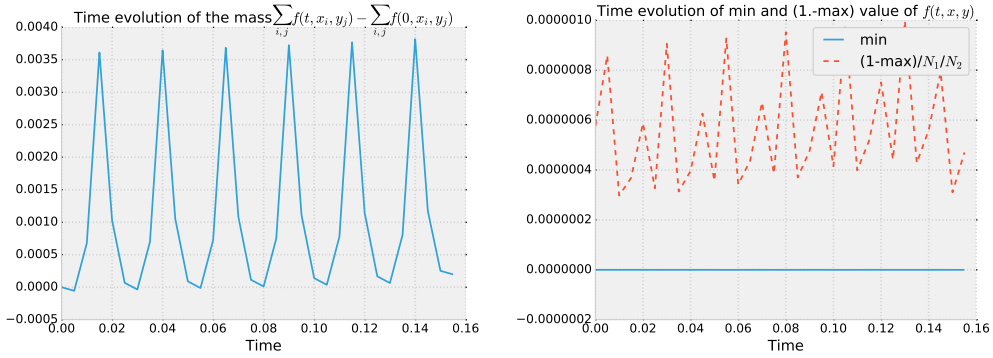
Figure 3.39: Test-case 10: Distribution function at time $t = 1.5$ with $N = 100$

maximum value and the error along time are indistinguishable in the figures. This is not unsurprising since, with this advection, the bulb does not go through the singular point nor the deformation. So our conclusion with this test-case is the same as with the previously mentioned test-case.

	$\varepsilon = 0.1$	$\varepsilon = 0.5$	$\varepsilon = 0.7$	$\varepsilon = 0.8$
L_∞	4.94577E-05	4.94577E-05	4.96367E-05	9.23684E-05
L_2	5.05076E-05	5.35099E-05	6.88522E-05	8.83304E-05

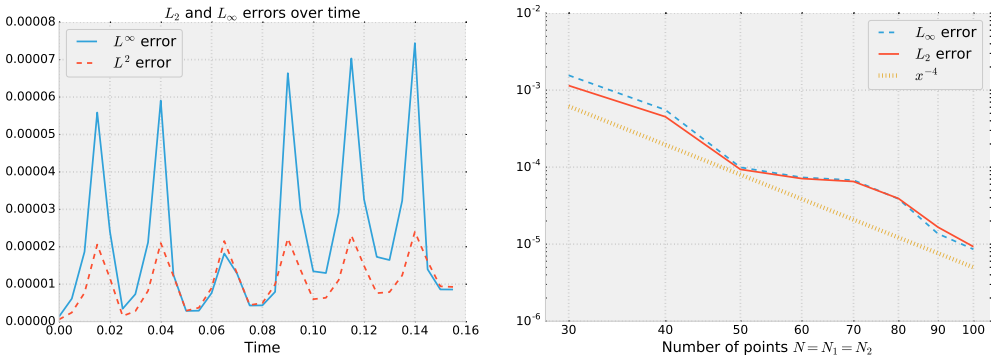
Table 3.3: Test-case 10: L_2 and L_∞ errors at t_max for different ε

For information, Table 3.3 shows the impact of different stretching coefficients. We notice that the minimum value of ε is zero, and the maximum is 1. We took



(a) Mass evolution over time **(b)** Minimal value evolution

Figure 3.40: Test-case 10: Time evolution of the mass and the minimal and maximal values, using $N = 100$ and $\Delta t = 1.5$



(a) Errors computed with $N = 100$ **(b)** Errors computed at time $t_{max} = 1.5$

Figure 3.41: Test-case 10: Evolution of L_2 and L_∞ errors over time (left) and number of cells (right)

four different values through this interval, and the results are all of the same order. There is a slight increase proportionally to ε, this is due to the fact that the deformation of the mesh starts to affect the mesh where the pulse goes through for this advection.

Test-case 11: Circular advection of Gaussian pulse through the critical points of the 5MP pinched disk

Similarly to what we did with Test-cases 7 and 10, we wish to recreate Test-case 8 –a circular advection of a Gaussian that goes through the singular points of the domain– although here the Gaussian pulse goes through the crown-disk interface and through the critical points (singular points stretched out). However, we wish

to compare the results of the stretching coefficient before comparing results. In Figure 3.42, we compare the resulting L_2 and L_∞ for different values of ε. The parameters of the simulation are: $t_{max} = 1.0$, $\Delta t = 0.005$, and we start the Gaussian pulse of width $\sigma = 0.004$ at $\mathbf{x}_c = (0.5, 0)$.

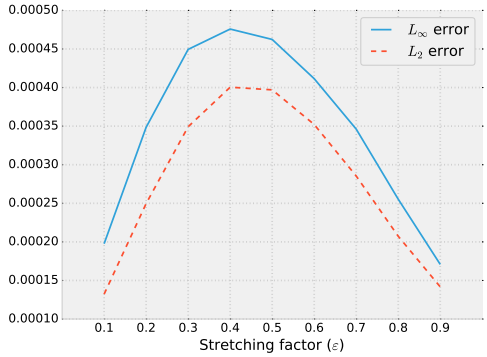
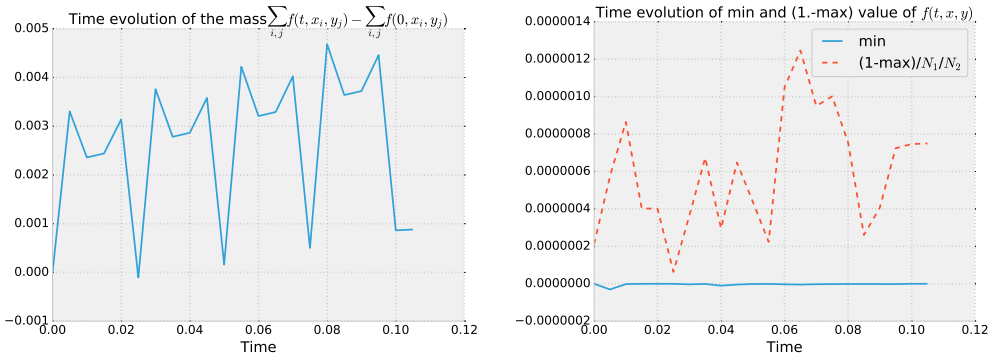


Figure 3.42: L_2 and L_∞ errors at $t_max = 1$ and $N = 50$ for different ε

We notice that the best results are at the extremes of the interval. However, when the coefficient tends to the minimum value, $\varepsilon \to 0$, the highest is the probably to find a numerically null jacobian for finer meshes. Thus, for the following test-case, we choose $\varepsilon = 0.9$.



(a) Mass evolution over time

(b) Minimal value evolution

Figure 3.43: Test-case 11: Time evolution of the mass and the minimal and maximal values, using $N = 100$ and $\Delta t = 1.5$

For Figures 3.43 and 3.45, we kept the same parameters except for the time step. Here $\Delta t = 0.0005$. Let us analyze the results: first of all, we notice that the peaks found in the mass evolution graphic (see Figure 3.43a) have two spikes instead of one as in all previous cases. This is probably due to the fact that we the stretched

geometry, the Gaussian has to go through twice as many patch interfaces than in the standard 5-patch disk. To illustrate this, we refer to Figure 3.44.

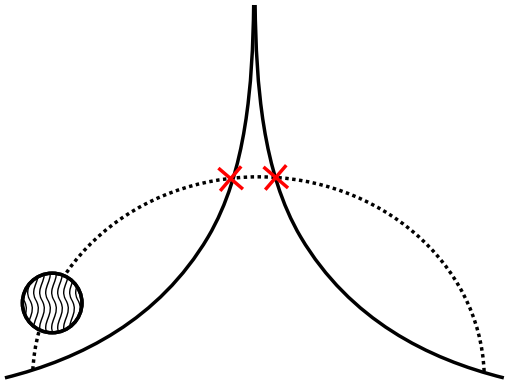
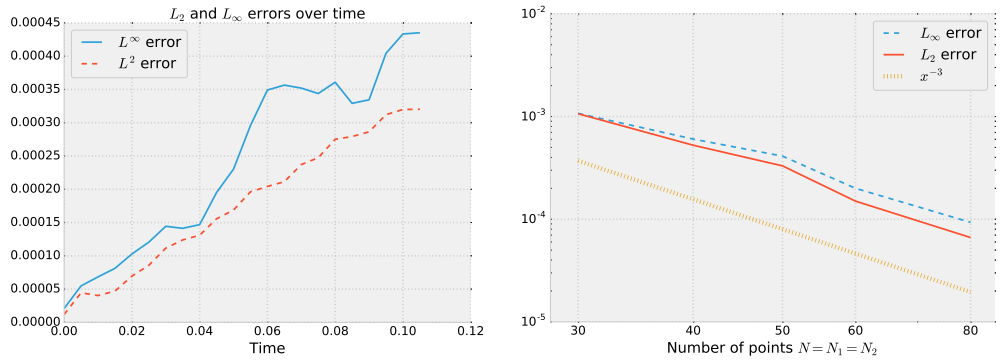


Figure 3.44: Sketch of Gaussian pulse trajectory around pinched domain

Additionally, even though the mass and the extremum values (see Figure 3.43b) are better conserved than on the un-pinched domain, the L_2 and L_∞ (Figure 3.45a) error are not.



(a) Errors computed with $N = 100$ **(b)** Errors computed at time $t_{max} = 1.5$

Figure 3.45: Test-case 11: Evolution of L_2 and L_∞ errors over time (left) and number of cells (right)

Nevertheless, the order of convergence is still three. We conclude that this geometry is better to avoid the singular points overall although the classical 5MP results is a satisfactory configuration with all the tools chosen. To study self-consistent simulations, for example a model coupled with a Poisson equation, this would require the implementation of a Poisson solver on multiple-patches. Since there is no problem of advection, and the Poisson equation can be solved in each patch (taking the right boundary conditions into account) this should be straightforward. Unfortunately, this type of problems are not studied here. However,

while exploring the Multi-patch Approach, we came upon a possible alternative to discretize the 2D poloidal plane. We explore this solution in the next Chapter.

4

The hexagonal mesh

While researching the Multi-patch Approach, we came across a discretization that would be the inspiration to the approach presented in this chapter. We called this grid the hexagonal mesh, or hex-mesh for short. The idea surged from the work of B. Scott and T. Ribeiro, and was attractive to us since it presented the possibility to map a logical mesh to a circle by a simple transformation and without any singular points. Most of the results in this Chapter appear in [MMPS16].

A regular triangular mesh: the hexagonal mesh

There are three kinds of regular paving 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 4.1).

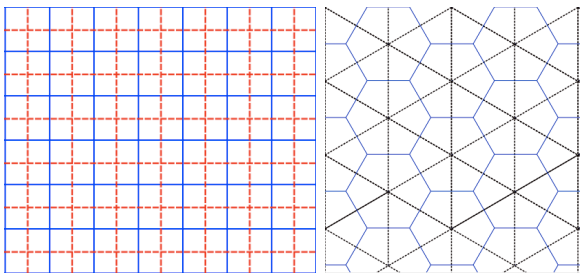


Figure 4.1: A square mesh and its dual (left), and a hexagonal mesh, in solid blue, and its dual mesh, the triangular tessellation, in dashed black (right).

Tiling a regular hexagon into triangles yields a mesh of equilateral triangles hav-

ing all the same area. Such a mesh was first introduced for numerical simulations in [SAM68]. An application to particle methods is proposed in [CL08]. 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. 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. Additionally, such a mesh has a structure with three privileged directions, and uniform steps in each direction, thus it is completely straightforward to localize 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 [CVDV08]. These splines have a hexagonal support and are invariant by translations along the three directions of the mesh.

In this Chapter, we focus on solving a guiding-center approximation of the 2D Vlasov Poisson system[GSR98] in such the hex-mesh. The model consists of a system of two equations: an advection equation and a Poisson equation. The first one is solved using the Semi-Lagrangian method. Thus, the first part is dedicated to 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. The interpolation method using Box-splines was presented in Equation (2.10); the second part, the back-tracing of the particles is presented in Chapter 4. Test-case 13 presents a finite difference Poisson solver adapted to the hexagonal mesh. Finally, in Test-case 13, we compare the results of the scheme using box-splines with the ones using Hermite finite elements (see [MMPS16]).

The BSL scheme on the hexagonal mesh

The guiding-center 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

$$\frac{\partial \rho}{\partial t} + \mathbf{A} \cdot \nabla_{\mathbf{x}} \rho(\mathbf{x}, t) = 0 \tag{4.1}$$

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. As in previous chapters, we apply the backward Semi-Lagrangian scheme to solve both the advection in a given field and the guiding center model.

Computing the origin of the characteristics

We consider the model Equation (4.1) on a 2D hexagonal domain, discretized with the hexagonal mesh. The points of the lattice are denoted $\mathbf{x} = (x_1, x_2)$. The distribution function $\rho(\mathbf{x}, t)$ is known on all grid points at the initial time $t = 0$. Let A_{x_1} and A_{x_2} be respectively the first and second components of $\mathbf{A}$. We proceed to apply the BSL method to the Vlasov Equation (4.1): 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

$$\begin{cases} \dfrac{d\mathbf{X}}{dt} &= \mathbf{A} \\ \mathbf{X}(s) &= \mathbf{x} \end{cases} \iff \begin{cases} \dfrac{d\mathbf{X_1}}{dt} &= A_{x_1} \\ \dfrac{d\mathbf{X_2}}{dt} &= A_{x_2} \\ X_1(s) &= x_1, \quad X_2(s) = x_2 \end{cases} \tag{4.2}$$

The solutions (X_1, X_2) of Equation (4.2) are called the characteristics associated with 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[FP15], to compute the origin of the characteristics,

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

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

Updating the distribution function

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

$$\rho(\mathbf{X}(t), t) = \rho(\mathbf{X}(s), s) = \rho(\mathbf{x}, s). \tag{4.3}$$

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

$$\rho^{n+1}(\mathbf{x}) = \rho^{n+1}(\mathbf{X}^{n+1}) = \rho^n(\mathbf{X}^n) \tag{4.4}$$

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 4.2). 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 (4.4) 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-spline coefficients, defined in Equation (2.18), are computed knowing $s[\mathbf{k}_i] = \rho^n(\mathbf{x}_i)$ (see Equation (2.10)), or the Hermite Finite-Element interpolation defined in [MMPS16], 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.

Localizing the characteristics' origins

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 Equation (4.2). Then

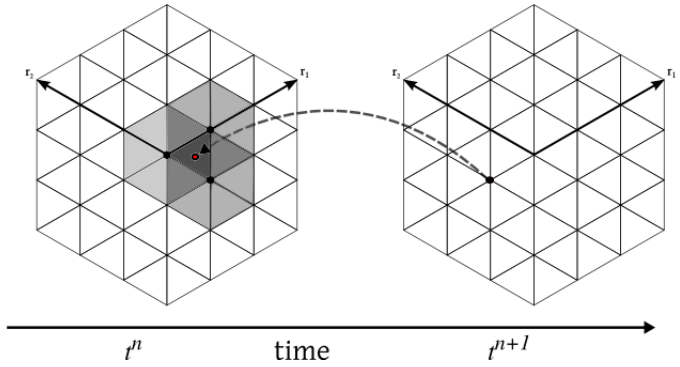


Figure 4.2: Semi-Lagrangian method: Tracing back characteristics.

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 Equation (2.1), and take the integer value. Denoting by (r_{ij}) the coefficients of the matrix $\mathbf{R}$, we get

$$\begin{cases} k_1 &= \left\lfloor \dfrac{r_{22}X_1 - r_{12}X_2}{r_{11}r_{22} - r_{12}r_{21}} \right\rfloor &= \left\lfloor \dfrac{1}{\sqrt{3}} (X_1 + X_2) \right\rfloor, \\ k_2 &= \left\lfloor \dfrac{-r_{21}X_1 + r_{11}X_2}{r_{11}r_{22} - r_{12}r_{21}} \right\rfloor &= \lfloor X_1 + X_2 \rfloor . \end{cases} \tag{4.5}$$

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.

Numerical Results

In this section, we will show the results obtained for the advection equation on the hexagonal mesh. The Test-cases were chosen as close as possible as the ones for the Multi-patch Approach (see Chapter 3). Additionally, we show the results with different degrees of Box-splines as well as different pre-filters.

Test-case 12: Short simulation of a constant advection model to compare pre-filters and degrees

Firstly, we want to narrow down the choice of pre-filter for the interpolation step. We chose a simple a short simulation, similar to Test-case 2, a linear advection of 0.1 along the x-axis of a Gaussian pulse. For the initial distribution function,

the pulse is centered at the origin of the a hexagonal mesh of radius $L = 1$. Furthermore, the pulse's amplitude is 1, and $\sigma = 0.04$. However, the simulation has a short maximum time, $t_{max} = 1$, with a time step of $\Delta t = 0.05$. For the interpolation, we chose to compare degree 1 and 2 for all four types of pre-filters previously mentioned (see Equation (2.10)).

	Degree	Number of cells N_c				Order
		10	20	40	80	
p_{FIR}	1	6.35-02	2.18E-02	6.07E-03	1.52E-03	2
	2	8.75E-03	2.46E-04	8.94E-06	4.52E-07	4
p_{IIR2}	1	2.02E-01	6.67E-02	2.44E-02	1.36E-02	1.3
	2	3.27E-01	9.24E-02	2.05E-02	1.69E-03	2.53
p_{IIR1}	1	2.03E-01	5.99E-02	1.53E-02	3.84E-03	1.91
	2	3.26E-01	9.43E-02	2.44E-02	6.15E-03	1.91
p_{int}	1	1.32E-01	3.67E-02	9.24E-03	2.31E-03	2.0
	2	3.25E-01	9.42E-02	2.44E-02	6.15E-03	2.0

Table 4.1: Convergence study of L_∞ errors for the quasi-interpolation method on the hexagonal meshes of sizes N_c using different pre-filters and Box-splines of degree 1 and 2

Since the scheme is based on a quasi-interpolation, the results obtained are worth studying even for such a short time period (as it would be even with a single time step). In Tables 4.1 and 4.2, we study the order of convergence with norm L_∞ and L_2 for Box-spline quasi-interpolations using different pre-filters. Let us discuss first Table 4.1, the L_∞ convergence study. We notice that for pre-filters p_{int} and p_{IIR2}, we obtained the results predicted by the literature ([CVDVU06, CVDV07]). The pre-filter p_{IIR2} also behaves as predicted, we obtain results of order $d - 1$, where d is the degree of the Box-splines. However the best results are found with pre-filter p_{FIR}. The results show a super-convergence that was not found by previous articles. We believe this phenomenon comes from the particularity of this test-case. Nevertheless, we show the results more in detail in the following test-case.

The L_2 convergence study results are inconclusive, with the exception of the pre-filter p_{FIR}. Overall, after running multiple advection test-cases with (slightly) different parameters, we concluded that pre-filters p_{int} and p_{IIR1} are constantly worse than the other two. Furthermore, with pre-filter p_{FIR} we obtained globally the best results, in order and precision. Thus, the next test-case uses this pre-filter. To see the same taste-case for other pre-filters we refer the reader to Appendix C.

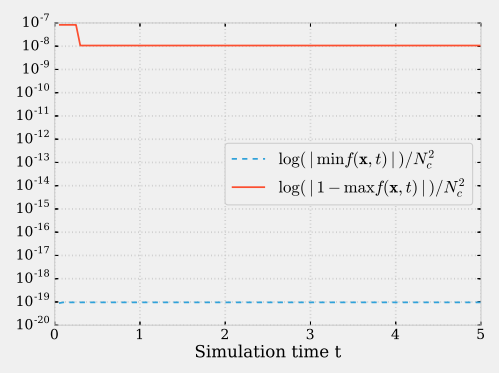
	Degree	Number of cells N_c				Order
		10	20	40	80	
p_{FIR}	1	3.55-01	2.12E-01	1.23E-01	6.59E-02	1
	2	5.17E-02	3.26E-03	2.71E-04	3.14E-05	3
p_{IIR2}	1	1.26E00	1.08E00	9.67E-01	1.01E-01	1.21
	2	2.21E-00	1.72E-00	8.99E-01	1.58E-01	1.26
p_{IIR1}	1	1.26E-00	9.58E-01	5.70E-01	3.06E-01	0.7
	2	2.21E-00	1.75E-00	1.06E-00	5.78E-01	0.7
p_{int}	1	7.66E-01	5.40E-01	3.14E-01	1.68E-01	0.72
	2	2.20E-00	1.75E-00	1.07E-00	0.58E-01	0.65

Table 4.2: Convergence study of L_2 errors for the quasi-interpolation method on the hexagonal meshes of sizes N_c using different pre-filters and Box-splines of degree 1 and 2

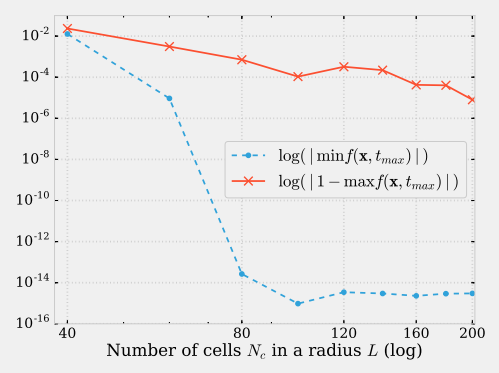
Test-case 13: Diagonal advection model and Box-spline quasi-interpolation with pre-filter P_{FIR}

We present the results found for the constant advection equation Equation (4.1). Here, we try to recreate as close as possible Test-case 5. Thus, we set a linear advection model on hex-mesh of radius $L = 1$, centered at the origin. Since the radius of the mesh is $L = 1$, and the number of cells N_c corresponds to the number of cells in a radius, the mesh step $\Delta x = 1/N_c$ is equivalent to the mesh step in the Multi-patch Approach. Indeed, the interpolations for the MPSL scheme are done in the logical domain of length 1 (independently of the physical domain) and with a mesh refinement of N. Thus, to have comparable mesh steps we set N_c to N. We advect the same distribution function as in Test-case 5: a Gaussian centered in $\mathbf{x}_c = (-0.4, -0.4)$ of variance $\sigma = 0.04$ and amplitude 1. The advection coefficient is $\mathbf{A} = (0.15, 0.15)$, while the parameters of the simulations are $t_{max} = 5$ with a time step $\Delta t = 0.05$. Finally, for the quasi-interpolation scheme, we used the pre-filter p_{FIR} based on the results of the previous test-case, and Box-splines of degree 1 and 2.

Let us first study the results for the bounds of the distribution function along time (Figure 4.3a) and for different mesh refinements (Figure 4.3b). Since, these values are conserved, we plot the results only for degree 2 Box-splines. We see that the conservation along time of the maximum value is close to machine error, and for the minimum we see the error constantly remains low. We should mention that,



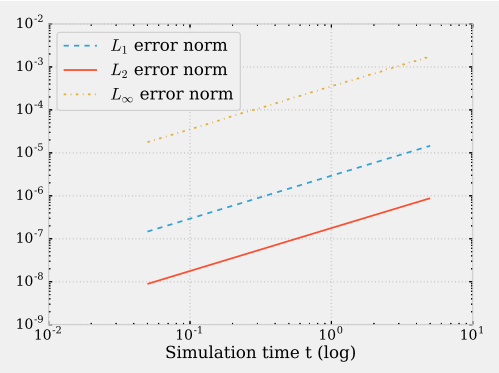
(a) Distribution bounds time evolution with $N_c = 100$



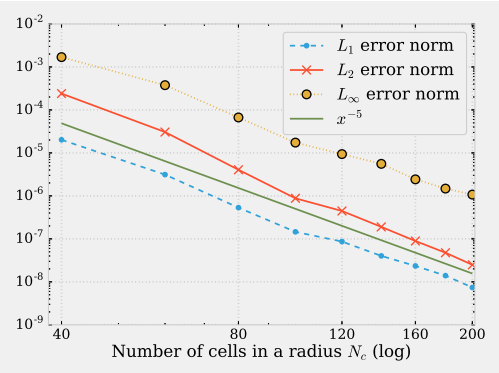
(b) Distribution bounds refinement evolution at $t_{max} = 5$

Figure 4.3: Test-case 13: Bounds of the distribution function over time (left) with $N_c = 100$ and number of cells (right) at $t = 5$

when working with Box-splines, these values are often hard to obtain. Actually, there are several studies on the stability of Box-splines [DBHR93, Kob97, KP09]. It follows that the evaluation we described in Chapter 2 preserves the maximum principle. However, we notice in Figure 4.3b that the mesh refinement is important.



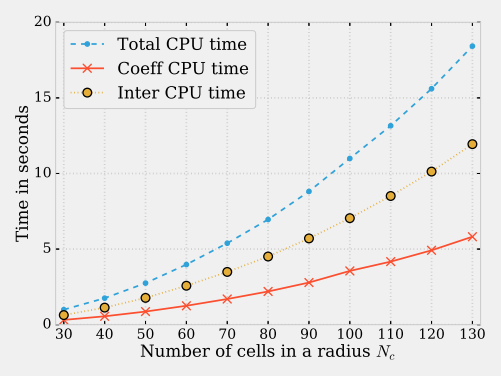
(a) Errors over time with $N_c = 100$



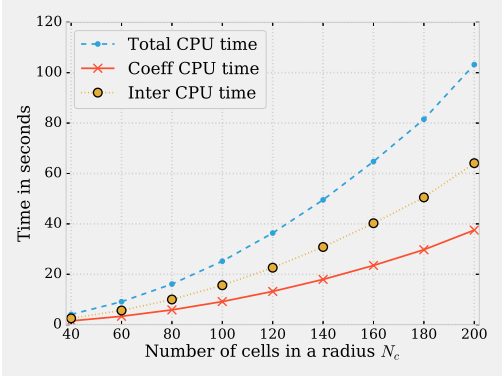
(b) Errors over mesh step at $t_{max} = 5$

Figure 4.4: Test-case 13: Evolution of L_1, L_2 and L_∞ errors over time (left) with $N_c = 100$ and over the number of cells (right) at $t = 5$ with degree 2 Box-splines

As in the Multi-patch Approach, we study the L_2 and L_∞, as well as the L_1 errors. We show here the results obtained for the degree 2 Box-splines, for the degree 1 results, see Appendix C. On the one hand, the convergence study (Figure 4.4) shows that the errors have a linear behavior along time as we expect from a linear advection. On the other hand, we see again the super convergence of the quasi-interpolation with the p_{FIR} pre-filter that we observed in the last test-case.



(a) CPU time used for simulations with degree 1 over mesh step



(b) CPU time used for simulations with degree 2 over mesh step

Figure 4.5: Test-case 13: CPU time used for quasi-interpolations with p_{FIR} on meshes of different refinements

Finally, in Figure 4.5, we compare the computational costs for degree 1 and degree 2 Box-splines. What we called "Coeff CPU time" is the time to compute the Box-spline coefficients (directly related to the type of pre-filter used), "Inter CPU time" is the time to perform the quasi-interpolation (independent of the type of pre-filter chosen), and the "Total CPU time" is simply the sum of the two. We notice that the splines of degree 2 use up to twice more of total CPU time than degree 1. However, the proportion of the time spent in computation of coefficients it is significantly higher (around 27% for $N_c = 130$ and degree= 1 versus 40% for $N_c = 200$ and degree 2 splines).

Globally, we found encouraging results with the quasi-interpolation method on advection models. We decide to explore further models on the mesh.

The Poisson finite-difference solver

When computing the origins of the characteristics for the Semi-Lagrangian scheme applied to 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 2.2a. 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 [CSSZ03]. 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 being 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 orders of precision at such little cost, we have used this fourth order scheme to compute ϕ.

Remark 8. *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 seven 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.*

General algorithm

Below, we summarize the full algorithm to compute the distribution function ρ^{n+1} solution of the guiding-center model Equation (4.1).

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 Equation (4.2), 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.

We should note that, here again, as for the Multi-patch Approach, the actual implementation was extensively discussed. The first decision was to add this code to the SeLaLib library –which was much more mature than it was at the beginning of this thesis but still far away from the actual version–. Thus, all the code related to the hexagonal mesh, the quasi-interpolation (and pre-filters) were implemented in SeLaLib. The finite difference Poisson solver, and the guiding-center simulations are also available therein.

However, the need for a more powerful Poisson solver motivated further developments. This time in the Django library[1]. Since this library was for Finite Element solvers, the decision was obvious. In order to reduce the implementations needed, everything that was related to the Semi-Lagrangian method stayed in SeLaLib, and an interface between the two libraries was implemented in SeLaLib.

We refer the author to Appendix B, for a list of the implementations and their locations.

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 scheme presented here as well as other Hermite elements (HCT-r, HCT-c, Z9, Z10, Ganev-Dimitrov and Mitchell) presented in [MMPS16]. 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.

Test-case 14: Circular advection

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

$$\partial_t f(x,y,t) + y\,\partial_x f(x,y,t) - x\,\partial_y f(x,y,t) = 0. \tag{4.6}$$

[1]This library started as a Finite Element code, also known as "the future Django". As for today, the library has evolved to such extent that its developer decided to divide it into several small libraries. The code mentioned here remains to be transported.

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:

$$f_0(x,y) = \exp\left(-\frac{1}{2}\left(\frac{(x-x_c)^2}{\sigma_x^2} + \frac{(y-y_c)^2}{\sigma_y^2}\right)\right), \tag{4.7}$$

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).

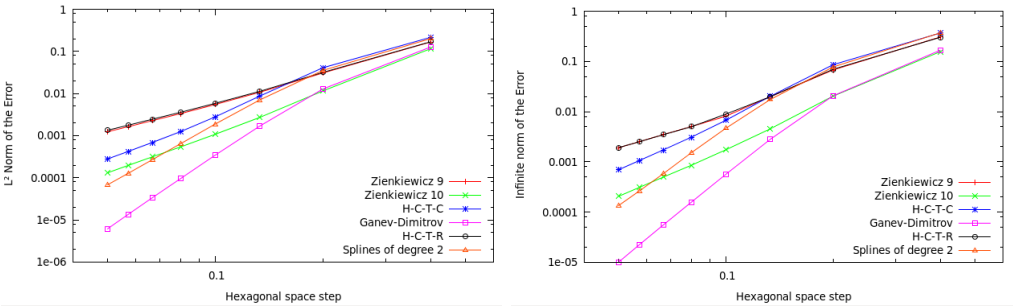


Figure 4.6: Test-case 14: Order of convergence with CFL $= 2$, with L_2 norm (left) and L_∞ norm (right)

In Figure 4.6, we plotted the L_2 and L_∞ norms for different space discretizations. We can see that for coarse meshes, all the methods are globally the same, with a slightly better accuracy for elements Zienkiewicz with 10 degrees of freedom (Z10 for short, based on polynomials of degree ≤ 3) and Ganev-Dimitrov (GaDi, 15 degrees of freedom and exact for polynomials of degree ≤ 4) [Ber94, GR14]. 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. The HCT-c and HCT-r elements are also defined in [Ber94, GR14] have respectively 12 and 9 degrees of freedom, and reproduce (resp.) polynomials of degree ≤ 2 and ≤ 3.

In Figure 4.7 (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 4.6), but we added the Mitchell elements (12 degrees of freedom, exact for polynomials of degree

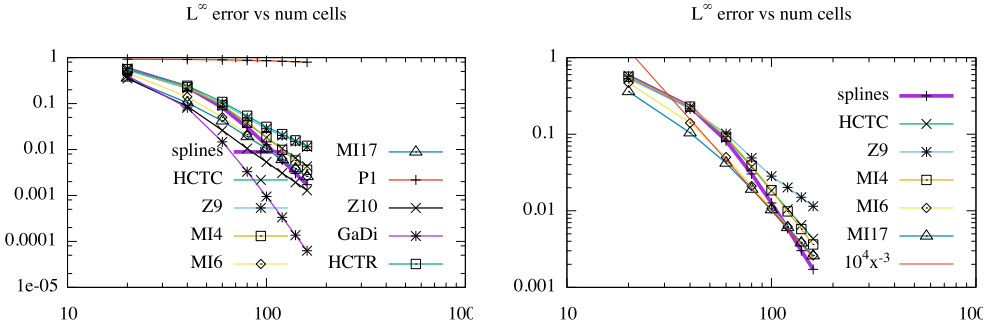


Figure 4.7: Test-case 14: Order of convergence (inclusion of Mitchell elements; CFL $= 0.25$) of all the methods (left) and only for schemes of order 3 (right).

≤ 4) [Mit73] 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. Other 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 4.7 (right), we selected the order 3 schemes (although we also added Zienkiewicz 9 (Z9), which is not exactly accurate to the order three but uses polynomials of degree ≤ 2), 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[CVDVU06]. On the other hand, the Mitchell elements behave favorably: MI4 is as accurate as HCT-c 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 [HMSS15]). The Z9 method is the least accurate, as expected. Note also that with respect to Figure 4.6, 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.

All previous figures study the convergence with respect to the hexagonal step or the number of cells N_c, thus in Figure 4.8 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 4.7 it is the best one since the beginning. This is probably due to the approximation of the normal derivatives. Furthermore, after a given mesh refinement, the box-spline interpolation seems to be the most accurate scheme.

In Figure 4.9, we can see that the performance converges quite quickly, for all

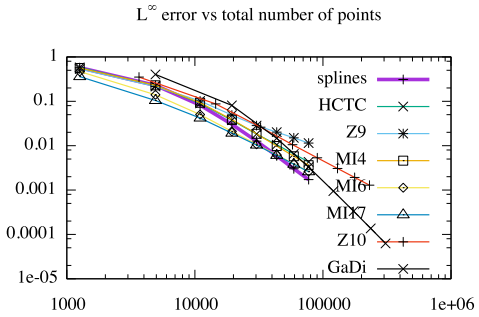


Figure 4.8: Test-case 14: Order of convergence (CFL $= 0.25$), with respect of number of points

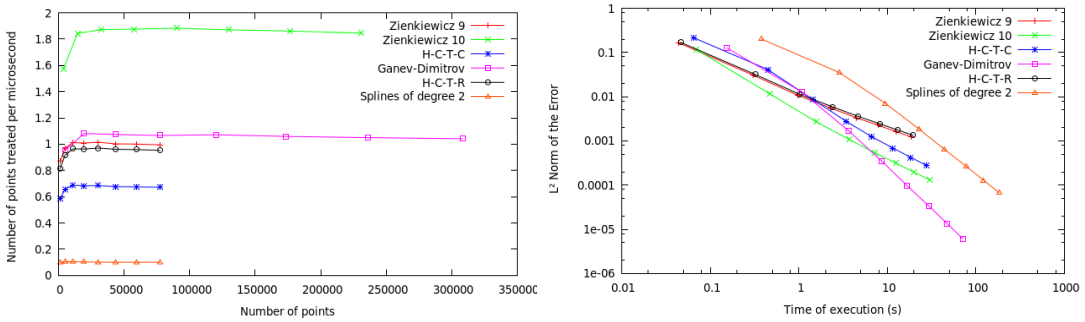


Figure 4.9: Test-case 14: Comparison of performances for the circular advection test case (CFL $= 2.$)

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.

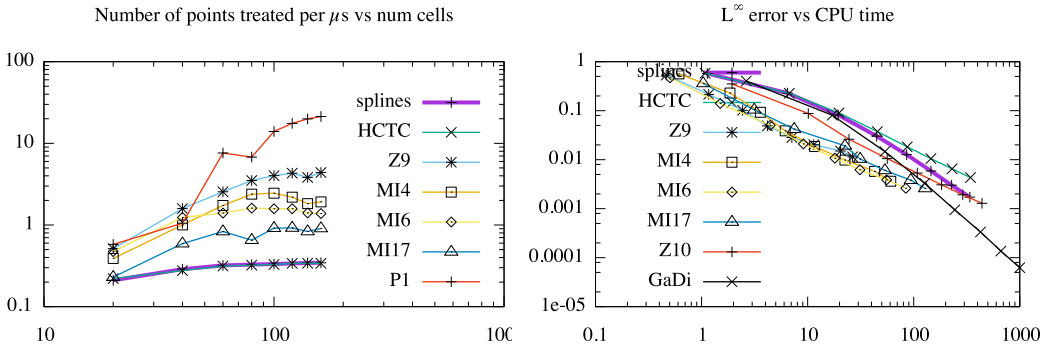


Figure 4.10: Test-case 14: Comparison of performances for the circular advection test case (inclusion of Mitchell elements; CFL $= 0.25$; on machine irmahpc2)

In Figure 4.10 (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 at 2.8GHz, 128 Go RAM, 2.4 To of disk. All the algorithms are not at the same

level of optimization: HCT-c 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 [MSM+13] for example. It is followed by the Z9 element, and the Mitchell elements which also need the computation of the mixed derivatives. Finally, splines and HCT-c have almost the same cost; note that previously, on Figure 4.9, splines were slower; HCT-c were also faster, but the runs were not done on the same computer.
We then represent on Figure 4.10 (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).

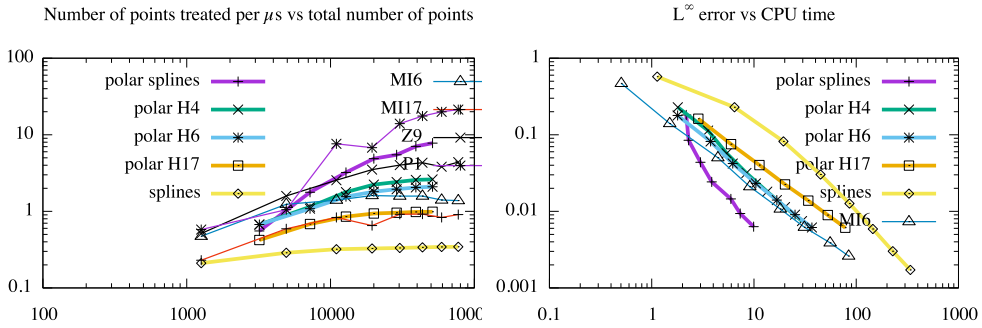


Figure 4.11: Test-case 14: Comparison of performances for the circular advection test case and comparison with classical interpolations on polar mesh; CFL = 0.25; on machine irmahpc2

Firstly, in Figure 4.11 (left), we study the performance with classical interpolation schemes on polar mesh (we use cubic splines and Hermite of degree p; see e.g. [HMSS15]). 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 4.11 (left), we compare again the number of points advected per μ-second. We see that in this polar setting, the classical cubic spline 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 4.11 (right). Cubic splines in polar geometry outperform 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.

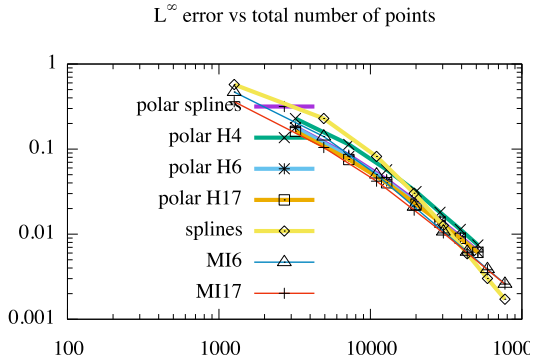


Figure 4.12: Test-case 14: Order of convergence (inclusion of Mitchell elements) and comparison with classical interpolation on polar mesh; CFL $= 0.25$

We then look Figure 4.12 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-spline interpolation is not the complete analog of cubic spline 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 remains quite similar between polar and hexagonal geometry, especially for the Hermite/Mitchell methods. Note also that the cubic spline method is almost the same as H6 (this is often the case, see [Ste14]).

Test-case 15: 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[FY14] 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

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + E_\perp \cdot \nabla_{\mathbf{x}} \rho(\mathbf{x}, t) = 0 & \text{(4.8a)} \\ -\Delta \phi = \nabla \cdot E = \rho(\mathbf{x}, t) & \text{(4.8b)} \end{cases}$$

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 [CGH+14]. The initial density is given by:

$$\rho_0(\mathbf{x}_\perp) = \begin{cases} (1 + \varepsilon \cos(\ell\theta)) \exp\left(-4(r - 6.5)^2\right), & \text{if } r^- \leq \sqrt{x^2 + y^2} \leq r^+. \\ 0, & \text{otherwise.} \end{cases} \tag{4.9}$$

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), $\theta = \text{atan2}(y, x)$. 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 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 $N_c = 160$.

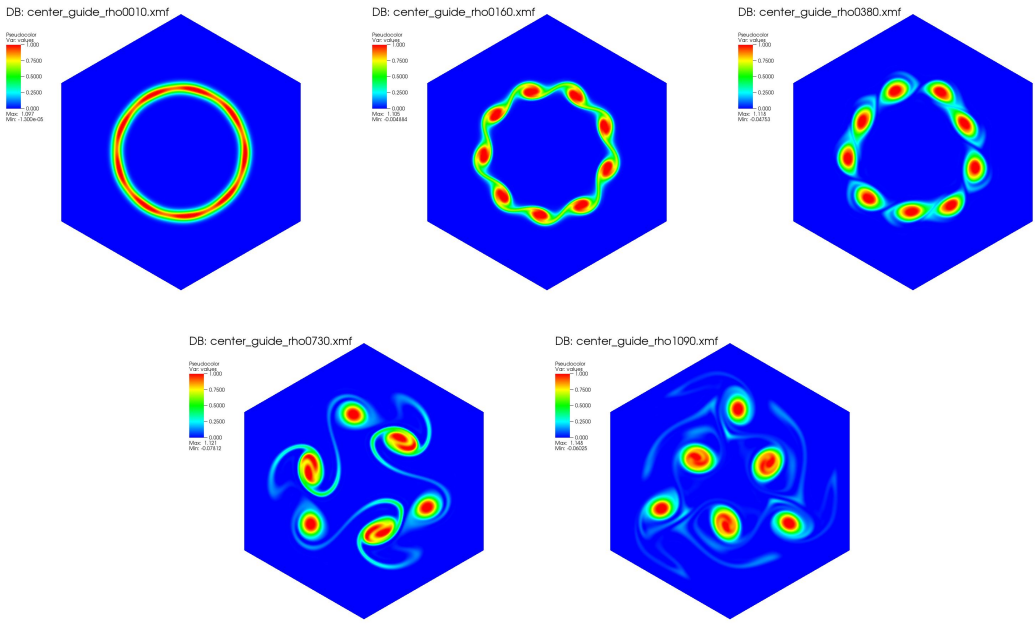


Figure 4.13: Test-case 15: Time evolution of the guiding-center model with $\varepsilon = 0.1$, at times = 1, 16, 38, 73 and 109

After playing with the parameters, we note that six vortices is the main mode. In fact, if we take $\ell \neq 6$, with ε small enough, we still see the mode six appear. With ε big enough, *i.e.* at least 0.1, the modes different from six 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 4.13, we can see the ninth mode turning into the sixth mode by fusion of vortices. This instability can be explained with Figure 4.14. 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[CGH+14]).

Remark 9. *When running the guiding-center model with test-case Equation (4.9), see Figure 4.13, no obvious differences are visible between the different interpolation*

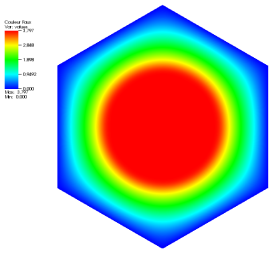


Figure 4.14: Test-case 15: Potential at time zero for the guiding-center simulation

methods, which makes the diagnostics all the more important to compare the results computed.

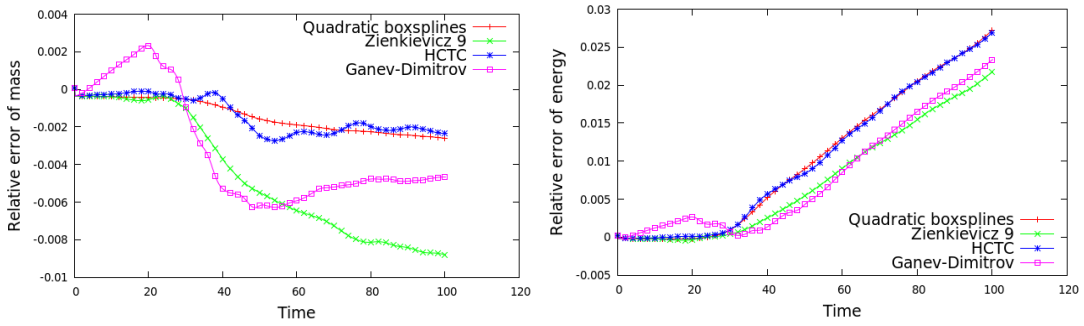


Figure 4.15: Test-case 15: Time evolution of the relative error of mass and energy

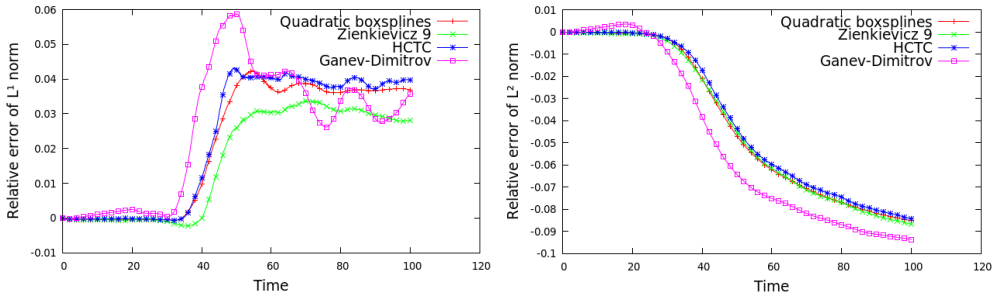


Figure 4.16: Test-case 15: Relative error of L^1 and L^2 norms

After comparison of the diagnostics in Figures 4.15, 4.16, and 4.17, we see that the various interpolation methods give close results overall. They are similar in terms of positivity conservation, especially 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

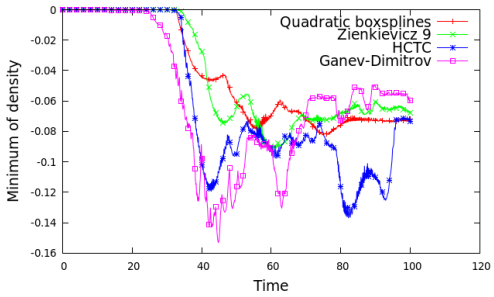


Figure 4.17: Test-case 15: Time evolution of the density's minimum

better the L^1 norm. Also we note that box-splines and the HCT-c element give very near results whichever the diagnostic considered.

In Figures 4.18, 4.19, and 4.20, we consider the same diagnostics as before, but we include a comparison with the Mitchell elements. Note that the Ganev Dimitrov does not have 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 4.18, we consider the case where $N_c = 128$, $\Delta t = 2^{-3}$. We then refine the mesh by 2, on Figure 4.19 and the time step by 2 on Figure 4.20, 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 not 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-spline method outperforms the other methods (probably due to their property of the partition of unity). HCT-c 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 4.19, 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

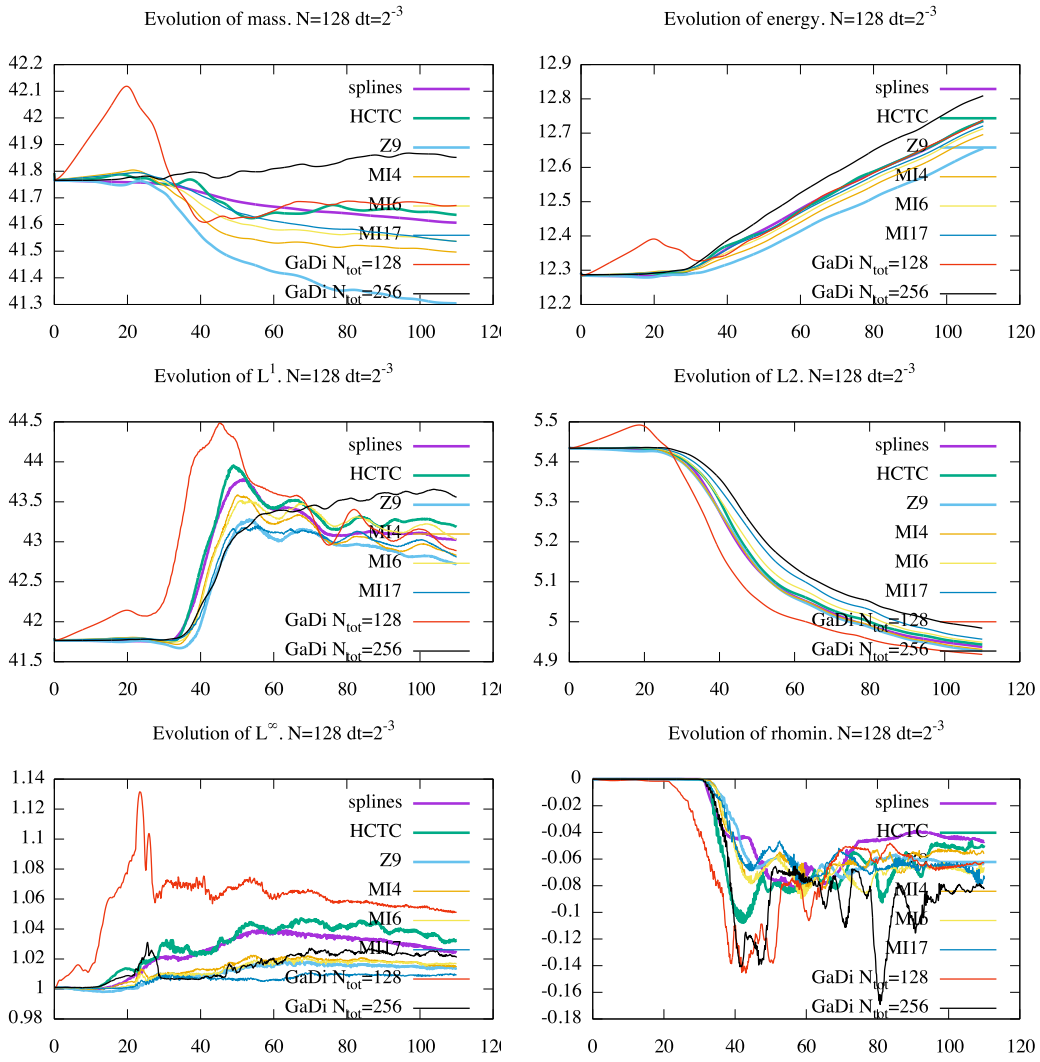


Figure 4.18: Test-case 15: Guiding-center case. Evolution of mass, energy, L^1, L^2, L^∞ and minimum density

is better than HCT-c 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. HCT-c, 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 points (see Figure 4.18). As previously mentioned, this is probably caused by the reconstruction of the normal derivatives. Concerning the L^∞ norm, the best results are obtained for Mitchell's elements, in particular MI17. Box-splines and HCT-c do not behave as well as the latter, and Z9 also behaves well. Finally, for the minimum density, HCT-c is one of the worst method. Box-

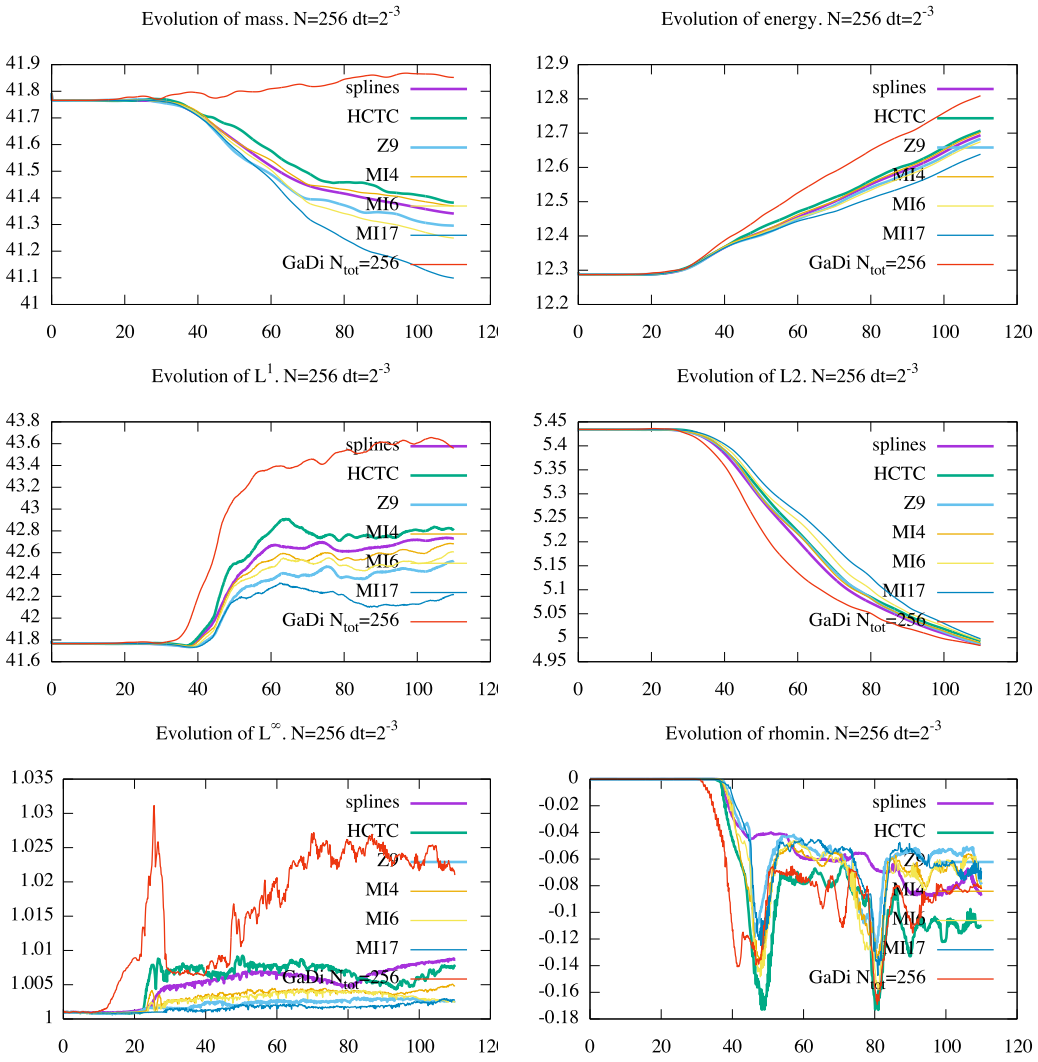


Figure 4.19: Test-case 15: Guiding-center case. Evolution of mass, energy, L^1, L^2, L^∞ and minimum density

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.

After this long study, we chose to pursue our research using Box-splines since they can be easily (even if at a high cost) increased in degree, and are as competitive as the other methods. They also have the advantage of being adaptable to other types of elements. However, since the Box-splines are not interpolant at the edges of the domain, in the next section we explore a solution to treat the boundary conditions.

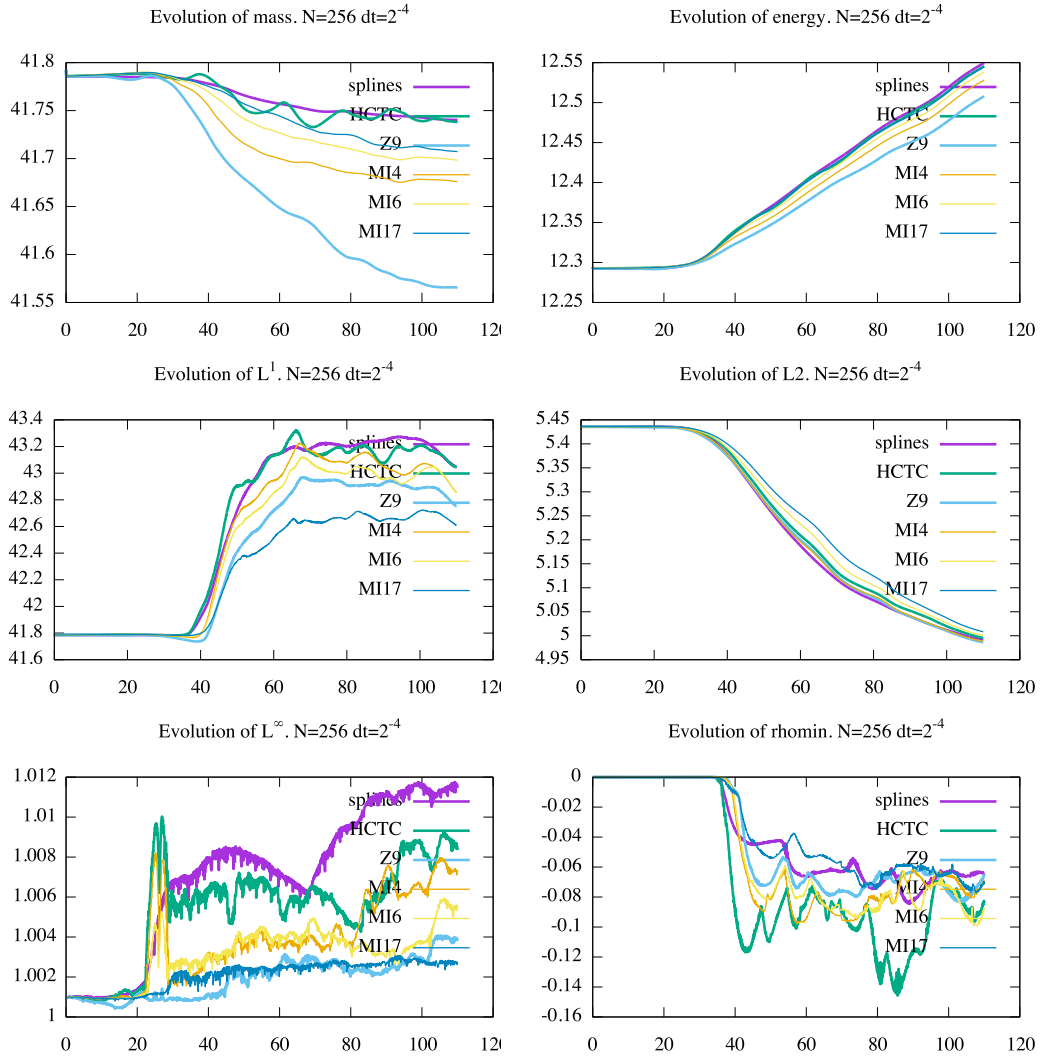


Figure 4.20: Test-case 15: Guiding-center case. Evolution of mass, energy, L^1, L^2, L^∞ and minimum density

Implementation of Nitsche's method on a hexagonal mesh

The Nitsche's method, well known in the Isogeometric community, deals with non-conforming meshes. It is also popularly used to handle Dirichlet boundary conditions in a weak form. It is often described as a combination of two other schemes that handle boundary conditions: the penalty and the Mortar's schemes. The first one usually leads to an ill-conditioned problem, depending heavily on the

penalty operator [TW05, ZTZ05]. Whereas the second one, the Mortar's method, generates complex solutions as the matrix of the discrete system is no longer positive definite [LM97, MAGS14]. Nitsche's method, introduced in 1971 [Nit71], has been proven to overcome these two problems [SLP12]. We want to apply it to the Poisson model on a hexagonal mesh as an example. This will, for example, allow us to perform the simulations of the last section, the guiding-center model, on a circular domain. Let us recall the Poisson equation in Cartesian coordinates

$$-\Delta\phi = \rho \qquad \text{in} \quad \Omega, \tag{4.10}$$

$$\phi = \phi^0 = g \qquad \text{on} \quad \Gamma_d, \tag{4.11}$$

$$\nabla\phi \cdot \mathbf{n} = h \qquad \text{on} \quad \Gamma_h. \tag{4.12}$$

We imposed here Neumann and Dirichlet boundary conditions. The first step is to write the variational formulation. We decided to use Embar's, Dolbow's and Harari's method [EDH10] to derive the variational form, but we recall that other methods exist [FMH04, GS03]. The problem Equation (4.10) to Equation (4.12) becomes: Find ϕ such that

$$a(w, \phi) \; = \; l(w) \tag{4.13}$$

with

$$a(w,\phi) = \int_\Omega \nabla w \cdot \nabla\phi \, \mathrm{d}\Omega - \int_{\Gamma_d} w(\nabla\phi \cdot \mathbf{n}) \, \mathrm{d}\Gamma - \int_{\Gamma_d} \phi(\nabla w \cdot \mathbf{n}) \, \mathrm{d}\Gamma + \alpha \int_{\Gamma_d} w\phi \, \mathrm{d}\Gamma$$
$$l(w) = \int_\Omega wf \, \mathrm{d}\Omega + \int_{\Gamma_h} wh \, \mathrm{d}\Gamma - \int_{\Gamma_d} g(\nabla w \cdot \mathbf{n}) \, \mathrm{d}\Gamma + \alpha \int_{\Gamma_d} wg \, \mathrm{d}\Gamma.$$

Where w is a test function. We can set, for example, $w = H^d$, a box-spline of degree d. Before discretizing the problem, let us introduce a coordinate transformation, F, whose Jacobian, $\mathbf{J}$, determinant is noted $|\mathbf{J}|$. The test function w, the electric potential ϕ and the density ρ, can be written as $\tilde{w}$, $\tilde{\phi}$, and $\tilde{\rho}$ in the new coordinate system. Furthermore, we recall the following properties:

$$\begin{aligned}
\nabla_{\mathbf{x}} f &= \mathbf{J}^{-T} \nabla_{\boldsymbol{\eta}} f \\
\mathbf{n}_{\mathbf{x}} &= \mathbf{J}^{-T} \mathbf{n}_{\boldsymbol{\eta}} \\
\int \mathrm{d}\Omega &= \int ||\mathbf{J}|| \, \mathrm{d}\mathcal{P} \\
\int \mathrm{d}\Gamma &= \int \sqrt{(\Delta x)^2 + (\Delta y)^2} \, \mathrm{d}\Gamma_{\mathcal{P}} = \int \Delta r \, \mathrm{d}\Gamma_{\mathcal{P}},
\end{aligned} \tag{4.14}$$

where f is any scalar function, such as w, ϕ, or ρ. Thus, the Equation (4.13) becomes

$$\tilde{a}(\tilde{w}, \tilde{\phi}) \; = \; \tilde{l}(\tilde{w}) \tag{4.15}$$

with

$$\tilde{a}(\tilde{w},\tilde{\phi}) = \int_{\mathcal{P}} \left(\mathbf{J}^{-T}\nabla\tilde{w}\right)\cdot\left(\mathbf{J}^{-T}\nabla\tilde{\phi}\right)||\mathbf{J}||\,\mathrm{d}\mathcal{P} - \int_{\partial\mathcal{P}_d}\tilde{w}\left(\left(\mathbf{J}^{-T}\nabla\tilde{\phi}\right)\cdot\left(\mathbf{J}^{-T}\mathbf{n}_{\boldsymbol{\eta}}\right)\right)||\mathbf{J}||\,\mathrm{d}\Gamma_{\mathcal{P}}$$
$$- \int_{\partial\mathcal{P}_d}\tilde{\phi}\left(\left(\mathbf{J}^{-T}\nabla\tilde{w}\right)\cdot\left(\mathbf{J}^{-T}\mathbf{n}_{\boldsymbol{\eta}}\right)\right)||\mathbf{J}||\,\mathrm{d}\Gamma_{\mathcal{P}} + \alpha\int_{\partial\mathcal{P}_d}\tilde{w}\,\tilde{\phi}\,||\mathbf{J}||\,\mathrm{d}\Gamma_{\mathcal{P}}$$
$$\tilde{l}(\tilde{w}) = \int_{\Omega}\tilde{w}\,\tilde{f}\,||\mathbf{J}||\,\mathrm{d}\mathcal{P} + \int_{\partial\mathcal{P}_h}\tilde{w}\,\tilde{h}\,||\mathbf{J}||\,\mathrm{d}\Gamma_{\mathcal{P}}$$
$$- \int_{\partial\mathcal{P}_d}\tilde{g}\left(\left(\mathbf{J}^{-T}\nabla\tilde{w}\right)\cdot\left(\mathbf{J}^{-T}\mathbf{n}_{\boldsymbol{\eta}}\right)\right)||\mathbf{J}||\,\mathrm{d}\Gamma_{\mathcal{P}} + \alpha\int_{\partial\mathcal{P}_d}\tilde{w}\,\tilde{g}\,||\mathbf{J}||\,\mathrm{d}\Gamma_{\mathcal{P}}.$$

Now, we use the following discretization for the solution ϕ.

$$\phi^h(\mathbf{x}_i) = \sum_{j=0}^{N_e(d)} c_j H_j^d(\mathbf{x}_i) \tag{4.16}$$

where $N_e(d)$ is the number of non-null splines H_j^d on a given element e for a given degree d, and c_j the coefficients associated to H_j^d. We recall H_j^d is the box-spline of degree d centered in $\mathbf{x}_j$. For simplicity of notations, we will assume here that only degree 1 box-splines are used. For higher degrees, the procedure remains the same with only a few parameters changing.

Coercivity and stability parameter evaluation

In this manuscript we won't detail the study on the coercivity of the scheme. Instead, we focus directly on the results found in [EDH10], and adapt them directly to our mesh. In fact, the previously cited studies show that ensuring the coercivity of the system Equation (4.13) is equivalent to ensuring the coercivity of

$$a(w^h,w^h) \geq (1-\varepsilon C)\left\|\nabla w^h\right\|^2 + \left(\alpha - \frac{1}{\varepsilon}\right)\left\|\nabla w^h\right\|_{\Gamma_d}^2 \tag{4.17}$$

where w^h is the discretized w and C is a mesh dependent constant such that

$$\left\|\nabla w^h\cdot\mathbf{n}\right\|_{\Gamma_d}^2 \leq C\left\|\nabla w^h\right\|^2. \tag{4.18}$$

The coercivity of Equation (4.17) is ensured when $\alpha > 1/\varepsilon$ with $\varepsilon < 1/C$. Furthermore, the definition of C will determine our stability. Let us define the following eigenvalue problem.

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x} \tag{4.19}$$

where the matrices A and B are defined by

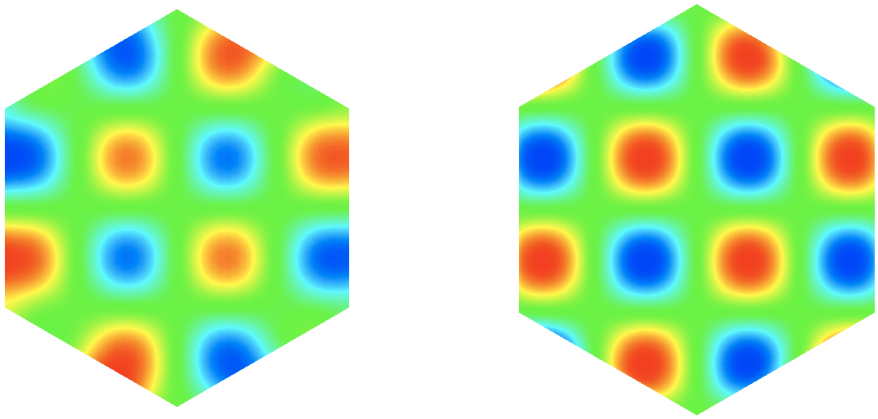
$$\begin{aligned}\mathbf{A}_{ij} &= \int_{\Gamma_d} (\nabla H_i^d \cdot \mathbf{n})(\nabla H_j^d \cdot \mathbf{n}) \mathrm{d}\Gamma \\ \mathbf{B}_{ij} &= \int_{\Omega} \nabla H_i^d \cdot \nabla H_j^d \mathrm{d}\Omega.\end{aligned}$$

Then, if C is equal to the maximum eigenvalue obtained from the system, the scheme is stable. Knowing this, and using the results of [DH09] –which show that $\alpha = 2C$ provides the best computational performance of the scheme–, we get $\alpha = 2\max(\mathbf{\Lambda})$. We need to rewrite Equation (4.19) in order to adapt it to the hexagonal mesh. Using the relations in Equation (4.14), it yields, for the matrix A,

$$\begin{aligned}\mathbf{A}_{ij} &= \int_{\Gamma_d} \left(\left(\mathbf{J}^{-T}\nabla H_i^d\right) \cdot \left(\mathbf{J}^{-T}\mathbf{n}_{\boldsymbol{\eta}}\right)\right) \left(\left(\mathbf{J}^{-T}\nabla H_j^d\right) \cdot \left(\mathbf{J}^{-T}\mathbf{n}_{\boldsymbol{\eta}}\right)\right) \mathrm{d}\Gamma \\ &= \int_{\Gamma_d} \left(\mathbf{J}^{-T}\nabla H_i^d\right)^T \left(\mathbf{J}^{-T}\mathbf{n}_{\boldsymbol{\eta}}\right) \left(\mathbf{J}^{-T}\nabla H_j^d\right)^T \left(\mathbf{J}^{-T}\mathbf{n}_{\boldsymbol{\eta}}\right) \Delta r \, \mathrm{d}\Gamma_{\mathcal{P}} \\ &= \Delta r \int_{\Gamma_d} (\nabla H_i^d)^T \mathbf{J}^{-1}\mathbf{J}^{-T}\mathbf{n}_{\boldsymbol{\eta}} (\nabla H_j^d)^T \mathbf{J}^{-1}\mathbf{J}^{-T}\mathbf{n}_{\boldsymbol{\eta}} \, \mathrm{d}\Gamma_{\mathcal{P}}.\end{aligned}$$

Following the same procedure, we get

$$\begin{aligned}\mathbf{B}_{ij} &= \int_{\mathcal{P}} \left(\mathbf{J}^{-T}\nabla H_i^d\right) \cdot \left(\mathbf{J}^{-T}\nabla H_j^d\right) |\mathbf{J}| \ \mathrm{d}\mathcal{P} \\ &= |\mathbf{J}| \int_{\mathcal{P}} (\nabla H_i^d)^T \mathbf{J}^{-1}\mathbf{J}^{-T}\nabla H_j^d \ \mathrm{d}\mathcal{P}.\end{aligned}$$



(a) before **(b)** after

Figure 4.21: Results of a Poisson equation before and after applying the Nitsche's method

In Figure 4.21, we applied a Poisson solver on a hex-mesh of $N = 32$ on which we defined a sinusoidal pavement. The results with null Dirichlet boundary conditions (before implementing the Nitsche method) are completely false as expected, whereas on the figure on the right we see that the boundary conditions are well treated.

For numerical reasons and time constraints it is impossible for us to assemble all the above-mentioned pieces together: the poisson solver on the hexagonal mesh and the Nitsche's method (Django) needs an interface for the input and output data coming from the interpolation method and the advection equation (SeLaLib). We strongly believe all the pieces have been presented in this work in order construct a complete guiding-center test.

Perspective: realistic poloidal planes

We previously mentioned that it would be easy to map the hexagon to a circle, and that it consists basically of a scaling, but we didn't go any further into details. In this section, we compute the mapping and notice its simplicity as well as its pitfalls.

Transformation of one triangle

Firstly, let us suppose that we are transforming only one triangle. We can take for example the triangle in our unit hexagon defined by the points $\mathbf{0}$, $\mathbf{1}$, and $\mathbf{2}$ (See Figure 4.22). We denote T_1 the triangle and $\mathbf{P_0}$, $\mathbf{P_1}$, and $\mathbf{P_2}$ its vertices. Only the points on the edge $\mathbf{P_2P_1}$ will be mapped, such that they are on the circle of center $\mathbf{P_0}$ and radius $r' = \|\mathbf{P_0P_2}\|$. Then, let $\mathbf{P}$ of coordinates (x, y) be a point on that edge, we want to compute the coordinates (x', y') of its projected point $\mathbf{P'}$. In polar coordinates the points $\mathbf{P}$ and $\mathbf{P'}$ have respectively the coordinates $(r\cos(\theta), r\sin(\theta))$ and $(r'\cos(\theta'), r'\sin(\theta'))$. First we notice that the points actually have the same angle, thus $\theta = \theta'$. Furthermore, we notice that the mapping is indeed a scaling, and that the scaling factor is nothing else than $k = \dfrac{r'}{r}$.

If we suppose that we know the coordinates of $\mathbf{P_2}$ then the computation of k is automatic ($r = \sqrt{x^2 + y^2}$ and $r' = y_{\mathbf{P}_2}$). But we suppose here that this information is not known. In fact, numerically, this will usually be the case, as we will be transforming the points not only on the external edges, but all other points inside the lattice as well. Nevertheless, finding the radius of the circumscribed circle, knowing only (x, y) is not a difficult task. We start by computing the slope of $\overline{\mathbf{P_1P_2}}$: we know the slope is the same regardless of the position in the hexagon, thus we can compute it using $\mathbf{P_1} = (\sqrt{3}/2, 1/2)$ and $\mathbf{P_2} = (0, 1)$. We get, s, the slope, is given by

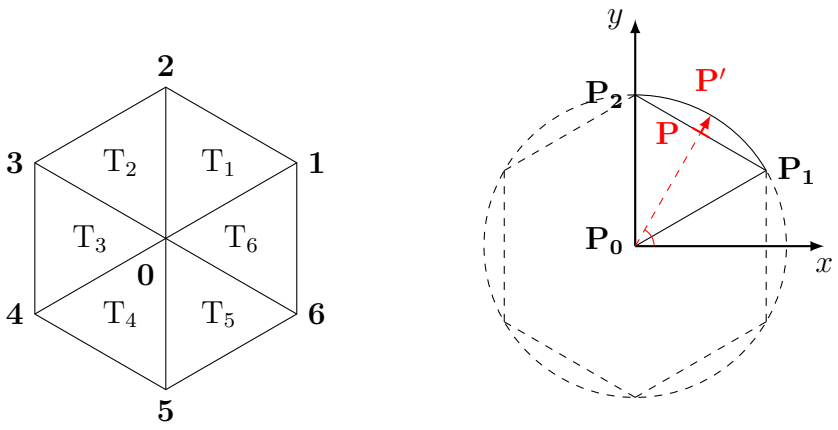


Figure 4.22: Test-case 15: Left: Element and node indexation on the unit hexagon. Right: Projection of a point $\mathbf{P}$ on T_1 into $\mathbf{P}'$ in the circumcircle of the hexagon.

$$s = \left(\frac{1}{2} - 1\right) \Big/ \frac{\sqrt{3}}{2} = \frac{-1}{\sqrt{3}}\,.$$

Thus, the equation of the line passing through $\mathbf{P_1P_2}$ is defined by

$$y_{\mathbf{P_2}} = y + \frac{1}{\sqrt{3}}x,$$

where $y_{\mathbf{P_2}} = r'$ is the unknown. We get the final formula: Given a point $\mathbf{P} \in \{\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_4, \mathrm{T}_5\}$, of coordinates (x, y), we can compute the coordinates of its projection $\mathbf{P}'$ of coordinates (x', y') into the circumscribed circle to the hexagon by the following formula,

$$\begin{cases} x' = k\,x \\ y' = k\,y \end{cases} \quad \text{with} \quad k = \frac{r'}{r} = \frac{|y| + \dfrac{|x|}{\sqrt{3}}}{\sqrt{x^2 + y^2}}. \tag{4.20}$$

The addition of the absolute values to x and y, makes this formula applicable to four of the six triangles of the main hexagon: $\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_4$, and T_5. For the triangles T_3 and T_6, the formula has to be changed completely as the slope of the edges are no longer $\pm 1/\sqrt{3}$. We proceed in the same manner to obtain a formula equivalent to Equation (4.20). For illustration we introduce Figure 4.23.

The unknown now is $r' = \|\mathbf{P_0P_1}\| = \sqrt{x^2_{\mathbf{P_1}} + y^2_{\mathbf{P_1}}}$. We realize that $x_{\mathbf{P_1}} = x$, abscissa of $\mathbf{P}$. Next, we compute the equation of the line passing through $\mathbf{P_0P_1}$. It is given by

$$\frac{1}{\sqrt{3}}x = y_{\mathbf{P_2}}.$$

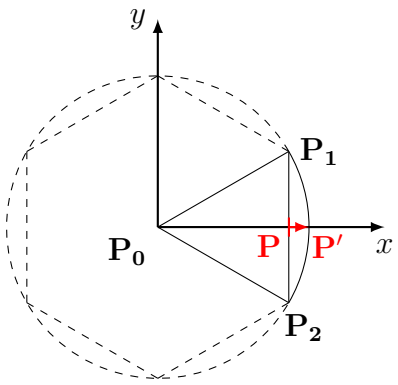


Figure 4.23: Test-case 15: Projection of a point $\mathbf{P}$ on T_6 into $\mathbf{P}'$ in the circumcircle of the hexagon.

Therefore, we have the value of $y_{\mathbf{P_2}}$ and we can compute r'. We obtain,

$$r' = \sqrt{x^2 + y_{\mathbf{P_1}}^2} = \sqrt{x^2 + x^2/3} = 2x/\sqrt{3}.$$

Thus, finally we get the formula for the mapping of P:

$$\begin{cases} x' = k\,x \\ y' = k\,y \end{cases} \quad \text{with} \quad k = \frac{r'}{r} = \frac{2x}{\sqrt{3(x^2+y^2)}}. \tag{4.21}$$

It is important to notice that transformation is only $\mathcal{C}^0$ continuous. This is caused mainly by the presence of the absolute value in Equation (4.20).

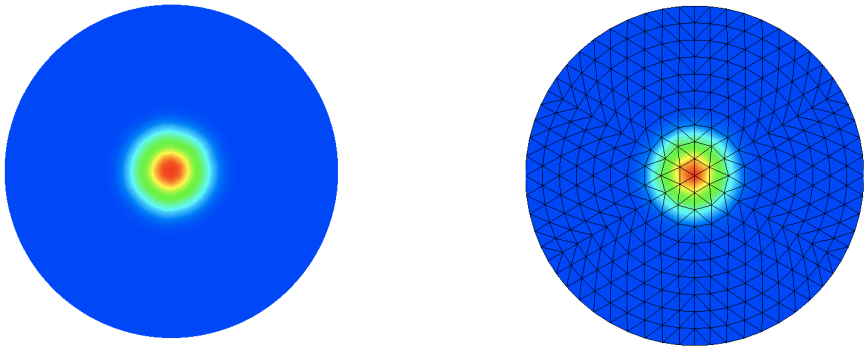


Figure 4.24: Mapped hex-mesh to a circle

We applied this transformation to a hex-mesh of $N_c = 10$. As a test-case, we applied a Gaussian to the origin of the mesh and solved a diffusion model. The results after 10 iterations are in Figure 4.24. In Figure 4.25, we applied

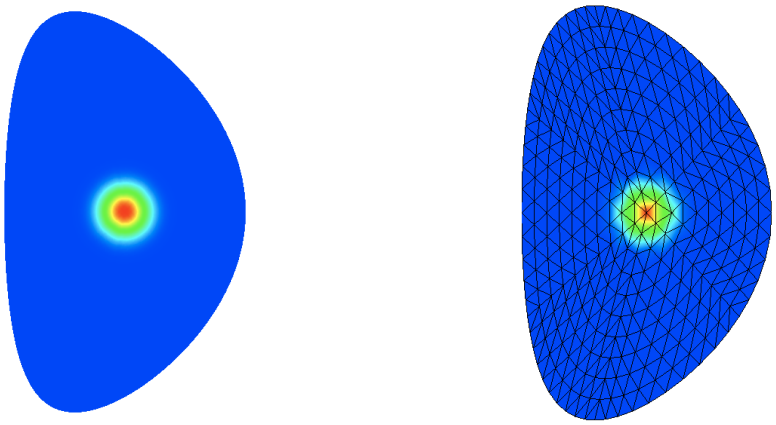


Figure 4.25: Mapped hex-mesh to Miller's equilibrium

the mapping described in [MCG+98], which describes the equilibria of an actual tokamak, and implemented the same test-case.

In the last example, it is noticeable that there is a problem due to the $\mathcal{C}^0$-continuity (derivatives not continuous) of the transformation. It is most noticeable at the three diagonals of the domain. However, to study the actual impact, we study the linear advection on this mapped domain.

Linear advection on the mapped hexagonal mesh

The linear advection equation in the mapped hexagonal domain (as a circle or a D-shape) is the same as the general curvilinear case, we recall Definition 8:

Given a coordinate transformation $F(\eta, \xi) = (F_1(\boldsymbol{\eta}), F_2(\boldsymbol{\eta})) = (x, y)$ of Jacobian $|\mathbf{J}|$, the constant coefficient advection equation in general coordinates reads

$$\frac{\partial \tilde{f}}{\partial t} + \tilde{\mathbf{A}} \cdot \nabla_{\boldsymbol{\eta}} \tilde{f} = 0 \tag{4.22}$$

where $\tilde{\mathbf{A}} = \frac{1}{|\mathbf{J}|} \overrightarrow{rot}_{\boldsymbol{\eta}} \tilde{\psi}$, $\tilde{\psi}(\eta, \xi) = a_1 F_2(\boldsymbol{\eta}) - a_2 F_1(\boldsymbol{\eta})$, with $(a_1 \ a_2) \in \mathbb{R}^2$ giving $\overrightarrow{rot}_{\boldsymbol{\eta}} \tilde{\psi}(\eta, \xi) = \begin{pmatrix} a_1 \partial_\xi F_2 - a_2 \partial_\xi F_1 \\ -a_1 \partial_\eta F_2 + a_2 \partial_\eta F_1 \end{pmatrix}$.

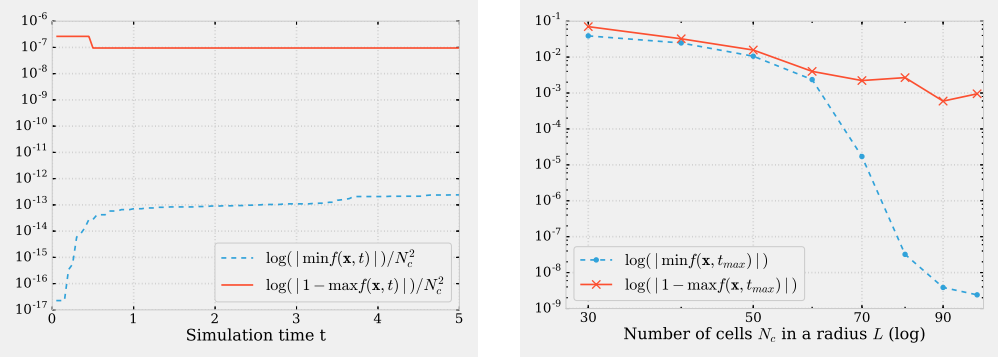
In this case, the coordinate transformation is either the hexagon $\longrightarrow$ circle transformation (given by Equations (4.20) and (4.21)), or the transformation hexagon $\longrightarrow$ D-shaped (given by Miller's equilibrium). Since the second transformation is actually a transformation hexagon $\longrightarrow$ circle $\longrightarrow$ D-shape, we study first the simple circle mapping.

Test-case 16: Constant advection of pulse through circular mapped hex-mesh

We intend to recreate Test-cases 5 and 9. We choose the distribution function

$$f_0(x,y) = \exp\left(-\frac{1}{2}\left(\frac{(x-0.5)^2}{0.04^2} + \frac{(y-0.5)^2}{0.04^2}\right)\right). \tag{4.23}$$

as in the previously mentioned test-cases. The studied model is a constant advection of the pulse (Equation (4.23)), with advection coefficient $\mathbf{A} = (0.5, 0.5)^T$. The simulation's parameters are $\Delta t = 0.05$ and $t_{max} = 5$. Based on the results of Test-case 13, we used Box-splines of degree 2, and based on the results of Test-case 13, we chose to use the p_{FIR} pre-filter. We mapped a hexagon of radius $L = 1$ to a circle of same radius using the coordinate transformation defined in Equations (4.20) and (4.21). The initial distribution parameters and the advection coefficients, give a simulation comparable to Test-cases 5 and 9. Furthermore, they assure that the Gaussian pulse goes through the mesh's origin, thus the results will show if the discontinuity of the transformation derivatives plays an important role.



(a) Distribution bounds time evolution with $N_c = 100$

(b) Distribution bounds refinement evolution at $t_{max} = 5$

Figure 4.26: Test-case 16: Bounds of the distribution function over time (left) with $N_c = 100$ and number of cells (right) at $t_{max} = 5$

First, let us see if the maximum principle is conserved over time and if the conservation improves when using a finer grid. In Figure 4.26, we note that for a refined mesh ($N_c = 100$) the minimum and maximum values of the distribution functions are conserved. For coarser grids, the minimal value is pretty well conserved, whereas the maximum value conservation improves with the refinement of the mesh. The maximum value has a worse conservation due to the dissipation of the distribution function through time (often seen when using the Semi-Lagrangian method).

In Figure 4.27, we plotted the difference between the analytical solution and the computed solution at time $t = 3.25$ (corresponding to the time when the Gaussian pulse goes through the center of the mesh).

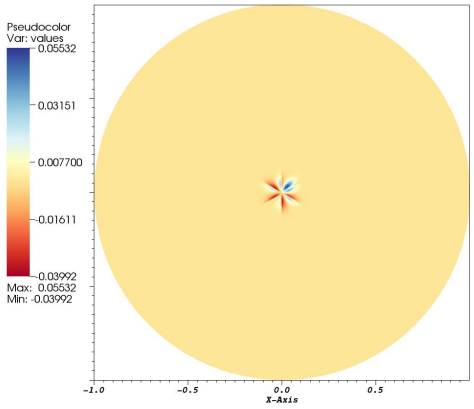
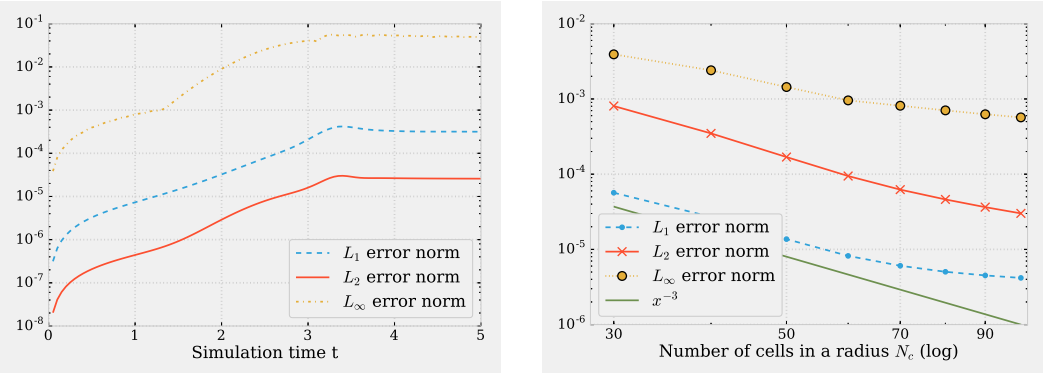


Figure 4.27: Test-case 16: Difference between the analytical and the computed solution of the distribution function at the critical time step $t = 5$

We can see that the $\mathcal{C}^0$ continuity of the derivatives along the six diagonals of the hexagon, introduces a point of high stress. This is also visible in Figure 4.28a, where we can see an obvious difference in the error norms evolution in time. After the Gaussian pulse passes through the origin, the error slope is not as important as before. What suggests that there might some major issues with more complicated models. However, we need more studies to confirm this.



(a) Errors over time with $N_c = 100$ **(b)** Errors over mesh step at $t_{max} = 5$

Figure 4.28: Test-case 16: Evolution of L_1, L_2 and L_∞ errors over time (left) with $N_c = 100$ and over number of cells (right) at $t = 5$ with degree 2 Box-splines

Lastly, in Figure 4.28b, we see that for coarse meshes ($N_c \leq 70$) the convergence rate is 3, while for finer meshes there is saturation. Probably this can be explained by the advection of the pulse through the origin of the domain. Nevertheless, yet again to confirm this, we need to simulate more complex models. This is the obvious next step in the study of this domain discretization.

5
Conclusion

We explored in this manuscript multiple approaches for the discretization of the 2D poloidal plane of fusion devices. The objective was to offer GYSELA with a solution that would represent the minimum effort to implement while still solving the singularity problem of the polar mesh and remaining flexible to treat more complex geometries. GYSELA is not the only code in the fusion community to face this problem. In fact, the Isogeometric analysis (IgA) was born from this kind of need. All our solutions are based on the principle of having a logical domain (patch) where we define our models, and a transformation that will map this space to the physical domain where we want the solution to be.

The models explored through this manuscript are simpler versions of the Vlasov equation eventually coupled to the Poisson equation. To solve the advective part, the Semi-Lagrangian scheme is used. This method consists of two steps: computing the characteristics feet and interpolating on those points.

The first family of solutions, heavily inspired by IgA, was the Multiple-Patch approach: the circular domain is divided into five patches. The four external patches, when combined, reconstruct exactly the current geometry of GYSELA, whereas the fifth, a square mapped to a disk, fills in the whole of the mesh. The main tools for this approach are the cubic splines: used for the interpolation method, used for the Semi-Lagrangian scheme, and for computing the slopes at patches' boundaries. We also presented with a naive and effective way of back-tracing the characteristics. The MPSL (Multi-patch Semi-Lagrangian) scheme was implemented from scratch in a Python library. We explored different configurations. At the end, two stood out, the classical 5-patch decomposition, and a modified version where the singular points are stretched outwards of the domain to eliminate the singularities. Both solutions, in our opinion, deserve further exploration.

The second solution explored lies on a different mesh. In fact, it is based on a hexagonal domain, that can be mapped to a circle without any singular points. This hexagonal domain is discretized using equilateral triangles. It yields that the back-tracing of the characteristics is practically automatic. We implemented a quasi-interpolation scheme on this mesh using Box-splines and their associated pre-filters. Moreover, two different Poisson solvers were implemented: a finite difference and a finite element solver. A guiding-center model was solved in this domain and compared to classical schemes in performance and accuracy. We implemented the Nitsche method in order to treat the boundary conditions. Finally, we implemented the coordinate transformations to map the domain to a circle or a tokamak-like domain.

We should notice that for the hexagonal mesh solution to be applied in GYSELA it needs to be linked to an external polar mesh (so the base code of GYSELA is kept as such). Actually, this can be achieved in a C^1 fashion. Although this was never tested in this work, we strongly believe this would be the perfect solution to our problem. In fact, doing so, the domain presents no singular points, we keep a polar crown, and we get accurate and efficient results.

A

Box-splines

From type-I Box-splines to our hexagonal Box-splines

The most common form of three directional Box-splines are the Type-I Box-splines[CL87, DBH83, Rat11], which generating vectors are the canonical base $\mathbf{e_1} = (1,0)^T$, $\mathbf{e_2} = (0,1)^T$, and their sum, $\mathbf{e_3} = (\mathbf{e_1} + \mathbf{e_2})$. Since many people in the IgA community work with this type of Box-splines[Sab02, LMS08, MPS11], we wish to link the type of Box-splines we use in this manuscript, with the type-I Box-splines.

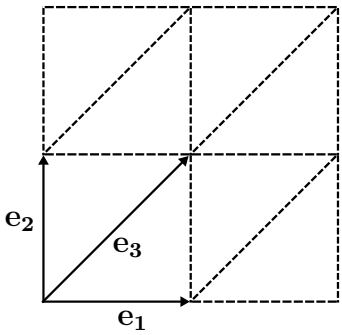


Figure A.1: Type-I Box-Splines: Generating vectors and sample mesh

The difference between the two types is simple: it lays on a linear modification of the geometry. Indeed, the generating matrix of the type-I Box-splines is $(\mathbf{e_1}\ \mathbf{e_2}\ \mathbf{e_3})$, while for our type of box splines is the $(\mathbf{r_1}\ \mathbf{r_2}\ \mathbf{r_3})$ matrix. Thus, the values of the functions differ by a constant, which comes from the definition of the Box-splines of degree 0 (see Definition 6).

We know that both $\{\mathbf{e_1}, \mathbf{e_2}\}$ and $\{\mathbf{r_1}, \mathbf{r_2}\}$ form a base of $\mathbb{R}^2$. Let $\mathbf{v} = v_1\,\mathbf{e_1} + v_2\,\mathbf{e_2}$, and $\mathbf{v'} = v_1\,\mathbf{r_1} + v_2\,\mathbf{r_2}$. X_i denotes the base formed by $(\mathbf{e_1}, \mathbf{e_2})$, and $\Xi_0 \cup X_i'$ the base of the hexagonal mesh. For the splines, we keep the notations used in this manuscript: B denotes the type-I Box-splines and χ the Box-splines in the hexagonal mesh. We obtain

$$\frac{1}{|\det(\Xi_0)|} B_{X_i}(\mathbf{v}) = \chi_{\Xi_0 \cup X_i'}(\mathbf{v'})$$

where Ξ_0, is the matrix for changing variables, here

$$\Xi_0 = \begin{pmatrix} \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix},$$

which gives in particular $\det(\Xi_0) = \frac{\sqrt{3}}{2}$.

An important property of the Box-splines is that they can be of different degrees on each direction (we do not use this property for the Box-splines in the hexagonal mesh). Let B^{d_1,d_2,d_3} and χ^{d_1,d_2,d_3} with $(d_1, d_2, d_3) \in \mathbb{N}^3$, be the Box-splines of type-I and on the hex-mesh, of degree d_1 in the first direction, d_2 on the second, and d_3 on the third. It yields

$$\frac{2}{\sqrt{3}} B^{d_1,d_2,d_3}(\mathbf{v}) = \chi^{d_1,d_2,d_3}(\mathbf{v'}) \tag{A.1}$$

This summarizes the relationship with the two types of splines. Thus, must properties proved for type-I Box-splines, are true for our hex-mesh Box-splines. In the next section, we show the main properties of the Box-splines necessary for the Isogeometric approach.

Important properties

Here is a list of properties that were essential to compute the results in the next section.

$$\partial_{\mathbf{e_n}} B(\mathbf{v}|X_i) = B(\mathbf{v}|X_i - \{\mathbf{e_n}\}) - B(\mathbf{v} - \mathbf{e_n}|X_i - \{\mathbf{e_n}\}) \tag{A.2}$$

$$\int_{\mathbb{R}^2} B^{n_1,n_2,n_3}(\mathbf{v} - \mathbf{v_0})\, B^{m_1,m_2,m_3}(\mathbf{v})\, d\mathbf{v} = B^{n_1+m_1,n_2+m_2,n_3+m_3}(n_1\mathbf{e_1} + n_2\mathbf{e_2} + n_3\mathbf{e_3} + \mathbf{v_0}) \tag{A.3}$$

$$\int_{\mathbb{R}^2} B(\mathbf{v}|X_n)\, f(v)\, d\mathbf{v} = \int_{[0,1]^n} f\left(\sum_{\mathbf{v} \in X_n} t_i \mathbf{v}\right)\, dt_1\, dt_2 ... dt_n \tag{A.4}$$

$$(n-d)\,B(\mathbf{v}|X_n) = \sum_{\zeta\in X_n} t_\zeta\, B(\mathbf{v}|X_n-\zeta) + (1-t_\zeta)\, B(\mathbf{v}-\zeta|X_n-\zeta) \tag{A.5}$$

where $B_M(\mathbf{v}) = B(\mathbf{v}|M)$ for a given matrix M and, for the first property, $n = 1, 2, 3$. In the third and fourth property, Equations (A.4) and (A.5), let $\mathbf{t} = X_n^t(X_nX_n^t)^{-1}\mathbf{v}$ where X_n is a matrix and t_ζ is the coordinate $\mathbf{t}$ at the ζ-th position in X_n. We note that, all the properties are true, if and only if, all the Box-splines on the right hand sides are defined and continuous on $\mathbf{v}$.

The first two properties follow by induction from the definition of the Box-splines. For the proof of the second property we refer the reader to Ratnani's thesis[Rat11]. The third property follows directly from the Box-splines definition. Finally, for the fourth property, the proof can be found in de Boor's book[DBHR93] (page 17).

Mass and Stiffness matrices

The Box-splines on the hexagonal mesh are used, among other things, as basis for the Finite Elements Methods. Thus, depending in the models, we will presented with the resolution of the Mass and/or Stiffness matrix. This part is dedicated to present the analytical and numerical solutions of such computations using the properties of the previous section.

Analytical computation

The Mass and Stiffness matrices are generally defined respectively by

$$\begin{aligned} M_{ij} &= \int \psi_i\, \psi_j \\ \Sigma_{ij} &= \int \nabla\psi_i \cdot \nabla\psi_j \end{aligned} \tag{A.6}$$

where ψ_i (respectively ψ_j) is the basis functions centered on the point of global index i (resp. j). In our context, the basis functions are actually the Box-splines of degree d on each direction ($\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$) of a hexagonal mesh of refinement $h = L/N_c$. While doing these computations, we realized that the easiest way to simplify these equations is by using the hexagonal coordinates which are, for a point of global index i, $(k_1^i, k_2^i) = (m, n)$. Respectively, for a point of global index j, we use the hexagonal coordinates $(k_1^j, k_2^j) = (k, l)$. We denote by $\chi_i^{d,d,d} = \chi_{m,n}^{d,d,d}$ (resp. $\chi_j^{d,d,d} = \chi_{k,l}^{d,d,d}$), the Box-spline of degree d on each direction, centered on

the point of global index i (resp. j). We use properties (A.2) and (A.3), with $|\mathbf{r_1}|=|\mathbf{r_2}|=1$ and $\mathbf{r_1}\cdot\mathbf{r_2}=-\frac{1}{2}$. It yields for the Mass Matrix

$$\begin{aligned} M_{mnkl} &= h^2 \int \chi^{d,d,d}_{m-k,n-l}(\mathbf{v})\,\chi^{d,d,d}_{0,0}(\mathbf{v})\,\mathrm{d}\mathbf{v} \\ &= \frac{2h^2}{\sqrt{3}}\chi^{2d,2d,2d}_{m-k,n-l}(2d\,\mathbf{r_3}). \end{aligned} \tag{A.7}$$

And for the Stiffness Matrix

$$\begin{aligned} \Sigma_{mnkl} = \int \Bigg[& \left(\chi^{d-1,d,d}_{m-k,n-l}(\mathbf{v}) - \chi^{d-1,d,d}_{m-k,n-l}(\mathbf{v}-\mathbf{r_1})\right)\left(\chi^{d-1,d,d}_{0,0}(\mathbf{v}) - \chi^{d-1,d,d}_{0,0}(\mathbf{v}-\mathbf{r_1})\right) \\ & + \left(\chi^{d,d-1,d}_{m-k,n-l}(\mathbf{v}) - \chi^{d,d-1,d}_{m-k,n-l}(\mathbf{v}-\mathbf{r_2})\right)\left(\chi^{d,d-1,d}_{0,0}(\mathbf{v}) - \chi^{d,d-1,d}_{0,0}(\mathbf{v}-\mathbf{r_2})\right) \\ & - \frac{1}{2}\Big[\left(\chi^{d-1,d,d}_{m-k,n-l}(\mathbf{v}) - \chi^{d-1,d,d}_{m-k,n-l}(\mathbf{v}-\mathbf{r_1})\right)\left(\chi^{d,d-1,d}_{0,0}(\mathbf{v}) - \chi^{d,d-1,d}_{0,0}(\mathbf{v}-\mathbf{r_2})\right) \\ & + \left(\chi^{d,d-1,d}_{m-k,n-l}(\mathbf{v}) - \chi^{d,d-1,d}_{m-k,n-l}(\mathbf{v}-\mathbf{r_2})\right)\left(\chi^{d-1,d,d}_{0,0}(\mathbf{v}) - \chi^{d-1,d,d}_{0,0}(\mathbf{v}-\mathbf{r_1})\right)\Big]\Bigg]\,dv \\ = \frac{2}{\sqrt{3}}\Big[& 2\,\chi^{2d-2,2d,2d}_{m-k,n-l}(2d\,\mathbf{r_3}-\mathbf{r_1}) - \chi^{2d-2,2d,2d}_{m-k+1,n-l}(2d\,\mathbf{r_3}-2\mathbf{r_1}) - \chi^{2d-2,2d,2d}_{m-k-1,n-l}(2d\,\mathbf{r_3}) \\ & + 2\,\chi^{2d,2d-2,2d}_{m-k,n-l}(2d\,\mathbf{r_3}-\mathbf{r_2}) - \chi^{2d,2d-2,2d}_{m-k,n-l}(2d\,\mathbf{r_3}-2\mathbf{r_2}) - \chi^{2d,2d-2,2d}_{m-k,n-l}(2d\,\mathbf{r_3}) \\ & - \frac{1}{2}\Big(\chi^{2d-1,2d-1,2d}_{m-k,n-l}(2d\,\mathbf{r_3}-\mathbf{r_1}) - \chi^{2d-1,2d-1,2d}_{m-k,n-l}(2d\,\mathbf{r_3}-\mathbf{r_1}-\mathbf{r_2}) \\ & - \chi^{2d-1,2d-1,2d}_{m-k,n-l}(2d\,\mathbf{r_3}) + \chi^{2d-1,2d-1,2d}_{m-k,n-l}(2d\,\mathbf{r_3}-\mathbf{r_2}) \\ & + \chi^{2d-1,2d-1,2d}_{m-k,n-l}(2d\,\mathbf{r_3}-\mathbf{r_2}) - \chi^{2d-1,2d-1,2d}_{m-k,n-l}(2d\,\mathbf{r_3}-\mathbf{r_2}-\mathbf{r_1}) \\ & - \chi^{2d-1,2d-1,2d}_{m-k,n-l}(2d\,\mathbf{r_3}) + \chi^{2d-1,2d-1,2d}_{m-k,n-l}(2d\,\mathbf{r_3}-\mathbf{r_1})\Big)\Big] \\ = -\frac{2}{\sqrt{3}}\Big(& \partial_{\mathbf{r_1},\mathbf{r_1}}\chi^{2d,2d,2d}_{m-k,n-l}(2d\,\mathbf{r_3}) + \partial_{\mathbf{r_2},\mathbf{r_2}}\chi^{2d,2d,2d}_{m-k,n-l}(2d\,\mathbf{r_3}) - \partial_{\mathbf{r_1},\mathbf{r_2}}\chi^{2d,2d,2d}_{i-k,j-l}(2d\,\mathbf{r_3})\Big)\,. \end{aligned} \tag{A.8}$$

While (A.7) and (A.8) are a general definition we can still reduce them by using the specificity of our hexagonal mesh. In [CVDV06], Condat and Van de Ville give the analytical expression of the Box-splines of contant degree (*i.e.* $d = d_1 = d_2 = d_3$, our case). However they use the following directions

$$\mathbf{u_1} = \begin{bmatrix} \frac{1}{2} \\ -\frac{\sqrt{3}}{2} \end{bmatrix} \qquad \mathbf{u_2} = \begin{bmatrix} \frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{bmatrix} \qquad \mathbf{u_3} = \begin{bmatrix} -1 \\ 0 \end{bmatrix},$$

which are equivalent to our generating matrix, by changing $\mathbf{u}_3$ into $-\mathbf{u}_3$ and applying a simple rotation. In fact, we notice that $\mathbf{r}_1 = (\mathbf{u}_{12}, \mathbf{u}_{11})^T$, $\mathbf{r}_2 = (\mathbf{u}_{22}, \mathbf{u}_{21})^T$

and $\mathbf{r}_3 = (-\mathbf{u}_{32}, -\mathbf{u}_{31})^T$. Let $\mathbf{v} = (x_1, x_2)^T$ a vector in cartesian coordinates, and (v_1, v_2) its coordinates in the $(\mathbf{r}_1, \mathbf{r}_2)$ base. We get $x_1 = \frac{\sqrt{3}(v_1 - v_2)}{2}$ and $x_2 = \frac{v_1 + v_2}{2}$. It follows

$$\begin{aligned}\chi^d(\mathbf{v} - d\,\mathbf{r_3}) = &\sum_{k_1,k_2=-d}^{d} \sum_{i=\max(k_1,k_2,0)}^{d+\min(k_1,k_2,0)} (-1)^{k_1+k_2+i} \binom{d}{i-k_1}\binom{d}{i-k_2}\binom{d}{i} \\ &\times \sum_{j=0}^{d-1} \binom{d-1+j}{j} \frac{1}{(2d-1+j)!\ (d-1-j)!} \\ &\times |v_2 - v_1 + k_1 - k_2|^{d-1-j} \left(\frac{v_1+v_2}{2} - \frac{k_1+k_2}{2} \right. \\ &\qquad \left. - \left|\frac{v_2 - v_1}{2} + \frac{k_1 - k_2}{2}\right|\right)_+^{2d-1+j}.\end{aligned} \tag{A.9}$$

This equation allows us to get a direct result for the Mass Matrix (by taking $\mathbf{v} = \mathbf{0}$). For the Stiffness Matrix, we have to derive this expression. We recall that Equation (A.9) is at least $\mathcal{C}^2$ for $d \geq 2$, given the Box-splines' properties. To lighten up the notations, we define

$$\begin{aligned} f_{k_1,k_2}(\mathbf{v}) &= |v_2 - v_1 + k_1 - k_2| \\ g_{k_1,k_2}(\mathbf{v}) &= (v_1 + v_2 - k_1 - k_2 - |v_2 - v_1 + k_1 - k_2|)_+ \\ \operatorname{sgn}(\mathbf{v}) &= \operatorname{sgn}(v_2 - v_1 + k_1 - k_2). \end{aligned}$$

Furthermore, since our expressions are $\mathcal{C}^2$, we get by continuity

$$\operatorname{sgn}(v_2 - v_1 + k_1 - k_2)^2 = 1.$$

Thus, the partial derivative along the first direction gives

$$\begin{aligned}\frac{\partial \chi^d}{\partial \mathbf{r_1}}(\mathbf{v} + d\,\mathbf{r_3}) = &\sum_{k_1,k_2=-d}^{d} \sum_{i=\max(k_1,k_2,0)}^{d+\min(k_1,k_2,0)} (-1)^{k_1+k_2+i} \binom{d}{i-k_1}\binom{d}{i-k_2}\binom{d}{i} \\ &\times \sum_{j=0}^{d-1} \binom{d-1+j}{j} \frac{1}{2^{2d-1+j}\ (2d-1+j)!\ (d-1-j)!} \\ &\times \big[-(d-1-j)\ \operatorname{sgn}(\mathbf{v}) f_{k_1,k_2}(\mathbf{v})^{d-2-j} g_{k_1,k_2}(\mathbf{v})^{2d-1+j} \\ &\quad + (2d-2+j)\,(1+\operatorname{sgn}(\mathbf{v}))\ f_{k_1,k_2}(\mathbf{v})^{d-1-j} g_{k_1,k_2}(\mathbf{v})^{2d-2+j}\big]\end{aligned}$$

and along the second derivative, we get

$$\frac{\partial \chi^d}{\partial \mathbf{r_2}}(\mathbf{v} + d\,\mathbf{r_3}) = \sum_{k_1,k_2=-d}^{d} \sum_{i=\max(k_1,k_2,0)}^{d+\min(k_1,k_2,0)} (-1)^{k_1+k_2+i} \binom{d}{i-k_1}\binom{d}{i-k_2}\binom{d}{i}$$
$$\times \sum_{j=0}^{d-1} \binom{d-1+j}{j} \frac{1}{2^{2d-1+j}\ (2d-1+j)!\ (d-1-j)!}$$
$$\times \big[(d-1-j)\ \operatorname{sgn}(\mathbf{v})\ f_{k_1,k_2}(\mathbf{v})^{d-2-j} g_{k_1,k_2}(\mathbf{v})^{2d-1+j}$$
$$+ (2d-2+j)\,(1-\operatorname{sgn}(\mathbf{v}))\ f_{k_1,k_2}(\mathbf{v})^{d-1-j} g_{k_1,k_2}(\mathbf{v})^{2d-2+j}\big]\,.$$

Now we can compute the compound derivatives,

$$\frac{\partial^2 \chi^d}{(\partial \mathbf{r_1})^2}(\mathbf{v} + d\,\mathbf{r_3}) = \sum_{k_1,k_2=-d}^{d} \sum_{i=\max(k_1,k_2,0)}^{d+\min(k_1,k_2,0)} (-1)^{k_1+k_2+i} \binom{d}{i-k_1}\binom{d}{i-k_2}\binom{d}{i}$$
$$\times \sum_{j=0}^{d-1} \binom{d-1+j}{j} \frac{1}{2^{2d-1+j}\ (2d-1+j)!\ (d-1-j)!}$$
$$\times \big[(d-1-j)(d-2-j)\, f_{k_1,k_2}(\mathbf{v})^{d-3-j}\, g_{k_1,k_2}(\mathbf{v})^{2d-1+j} - 2\,(2d$$
$$-2+j)\,(d-1-j)\,(\operatorname{sgn}(\mathbf{v})+1)\ f_{k_1,k_2}(\mathbf{v})^{d-2-j} g_{k_1,k_2}(\mathbf{v})^{2d-2+j}$$
$$+ (2d-2$$
$$+j)\,(2d-3+j)\,(1+\operatorname{sgn}(\mathbf{v}))^2\ f_{k_1,k_2}(\mathbf{v})^{d-1-j} g_{k_1,k_2}(\mathbf{v})^{2d-3+j}\big]$$

$$\frac{\partial^2 \chi^d}{(\partial \mathbf{r_1})(\partial \mathbf{r_2})}(\mathbf{v} + d\,\mathbf{r_3}) = \sum_{k_1,k_2=-d}^{d} \sum_{i=\max(k_1,k_2,0)}^{d+\min(k_1,k_2,0)} (-1)^{k_1+k_2+i} \binom{d}{i-k_1}\binom{d}{i-k_2}\binom{d}{i}$$
$$\times \sum_{j=0}^{d-1} \binom{d-1+j}{j} \frac{1}{2^{2d-1+j}\ (2d-1+j)!\ (n-1-j)!}$$
$$\times \big[-(d-1-j)(d-2-j)\ \operatorname{sgn}(v_2-v_1$$
$$+k_1-k_2)\ f_{k_1,k_2}(\mathbf{v})^{d-3-j} g_{k_1,k_2}(\mathbf{v})^{2d-1+j} - (d-1-j)\,(2d$$
$$-1+j)\,(\operatorname{sgn}(\mathbf{v})-1)\ f_{k_1,k_2}(\mathbf{v})^{d-2-j} g_{k_1,k_2}(\mathbf{v})^{2d-2+j} + (d-1$$
$$-j)\,(2d-1+j)\,(\operatorname{sgn}(\mathbf{v})+1)\ f_{k_1,k_2}(\mathbf{v})^{d-2-j} g_{k_1,k_2}(\mathbf{v})^{2d-2+j}$$
$$+(1-\operatorname{sgn}(\mathbf{v}))\,(1+\operatorname{sgn}(\mathbf{v}))\ f_{k_1,k_2}(\mathbf{v})^{d-1-j} g_{k_1,k_2}(\mathbf{v})^{2d-3+j}\big]$$

$$\frac{\partial^2\chi^d}{(\partial\mathbf{r_1})(\partial\mathbf{r_2})}(\mathbf{v}+d\,\mathbf{r_3}) = \sum_{k_1,k_2=-d}^{d}\sum_{i=\max(k_1,k_2,0)}^{d+\min(k_1,k_2,0)}(-1)^{k_1+k_2+i}\binom{d}{i-k_1}\binom{d}{i-k_2}\binom{d}{i}$$
$$\times\sum_{j=0}^{d-1}\binom{d-1+j}{j}\frac{1}{2^{2d-1+j}\ (2d-1+j)!\ (d-1-j)!}$$
$$\times\big[-(d-1-j)(d-2-j)\ \mathrm{sgn}\,(\mathbf{v})\ f_{k_1,k_2}(\mathbf{v})^{d-3-j}g_{k_1,k_2}(\mathbf{v})^{2d-1+j}$$
$$+2\,(d-1-j)\,(2d-1+j)\ f_{k_1,k_2}(\mathbf{v})^{d-2-j}g_{k_1,k_2}(\mathbf{v})^{2d-2+j}\big]$$

$$\frac{\partial^2\chi^d}{(\partial\mathbf{r_2})^2}(\mathbf{v}+d\,\mathbf{r_3}) = \sum_{k_1,k_2=-d}^{d}\sum_{i=\max(k_1,k_2,0)}^{d+\min(k_1,k_2,0)}(-1)^{k_1+k_2+i}\binom{d}{i-k_1}\binom{d}{i-k_2}\binom{d}{i}$$
$$\times\sum_{j=0}^{d-1}\binom{d-1+j}{j}\frac{1}{2^{2d-1+j}\ (2d-1+j)!\ (d-1-j)!}$$
$$\times\big[(d-1-j)(d-2-j)\ f_{k_1,k_2}(\mathbf{v})^{d-3-j}g_{k_1,k_2}(\mathbf{v})^{2d-1+j}+2\ (2d-2$$
$$+j)\,(d-1-j)\,(\mathrm{sgn}\,(\mathbf{v})-1)\ f_{k_1,k_2}(\mathbf{v})^{d-2-j}g_{k_1,k_2}(\mathbf{v})^{2d-2+j}+(2d$$
$$-2+j)\,(2d-3+j)\,(1-\mathrm{sgn}\,(\mathbf{v}))^2\ f_{k_1,k_2}(\mathbf{v})^{d-1-j}g_{k_1,k_2}(\mathbf{v})^{2d-3+j}\big]$$

Let $\mathbf{C}^d(j) = 1/\big(2^{2d-1+j}\ (2d-1+j)!\ (d-1-j)!\big)$, we can now compute the value of the Stiffness Matrix since we get the following definition

$$\frac{\partial^2\chi^d}{\partial\mathbf{r}_1^2}(\mathbf{v}+d\,\mathbf{r}_3)+\frac{\partial^2\chi^d}{\partial\mathbf{r}_2^2}(\mathbf{v}+d\,\mathbf{r}_3)-\frac{\partial^2\chi^d}{\partial\mathbf{r}_1\partial\mathbf{r}_2}(\mathbf{v}+d\,\mathbf{r}_3)$$
$$=\sum_{k_1,k_2=-d}^{d}\sum_{i=\max(k_1,k_2,0)}^{d+\min(k_1,k_2,0)}(-1)^{k_1+k_2+i}\binom{d}{i-k_1}\binom{d}{i-k_2}\binom{d}{i}\sum_{j=0}^{d-1}\binom{d-1+j}{j}\mathbf{C}^d(j)$$
$$\times\big(3(d-1-j)(d-2-j)f_{k_1,k_2}(\mathbf{v})^{d-3-j}g_{k_1,k_2}(\mathbf{v})^{2d-1+j}$$
$$-6\,(2d-2+j)\,(d-1-j)\,f_{k_1,k_2}(\mathbf{v})^{d-2-j}g_{k_1,k_2}(\mathbf{v})^{2d-2+j}$$
$$+4\,(2d-2+j)\,(2d-3+j)f_{k_1,k_2}(\mathbf{v})^{d-1-j}g_{k_1,k_2}(\mathbf{v})^{2d-3+j}\big)\,. \tag{A.10}$$

By multiplying by $-\frac{2}{\sqrt{3}}$ and taking v such that $(v_1, v_2) \in \mathbb{N}^2$, we get the value of the Stiffness Matrix.

Python algorithms

In this section we give the python codes to compute the Mass and Stiffness Matrices based on the results of the last section.

Mass Matrix

We use Equation (A.9) to compute $M_{mnkl}\sqrt{3}/2h^2 = \chi^{d,d,d}_{m-k,n-l}(2d\mathbf{r}_3) = \chi^{d,d,d}(u_0 + d, v_0 + d)$. Since u_0 and v_0 are integers, we can use the function *Fraction*, which gives the exact value.

```
def mass_matrix(u0,v0,d):
    """
        Computation of \chi_{d,d,d} (u0 + d, v0 + d)
    """
    u = np.copy(u0)
    v = np.copy(v0)
    x = u + v
    sqrt3y = v - u
    x = -dbs(x)
    sqrt3y = abs(sqrt3y)
    u = (x - sqrt3y)/2
    v = (x + sqrt3y)/2

    id=np.nonzero(v>0)
    v[id]=-v[id]
    u[id]=u[id]+v[id]

    id=np.nonzero(v>u/2)
    v[id]=u[id]-v[id]
    u = u.astype(int)
    v = v.astype(int)

    val=np.zeros(u.shape, dtype=Fraction)
    print(val)

    for K in range(-d, int(ceil(np.max(u)))):
        K0=Fraction(K)
        for L in range(-n, int(ceil(np.max(v)))):
            L0=Fraction(L)
            for i in range(0, min(d+K, d+L)+1):
                coeff=Fraction((-1)**(K+L+i)\
                          *binomialCoefficient(d,i-K)\
                          *binomialCoefficient(d,i-L)\
                          *binomialCoefficient(d,i))
                for j in range(0,d):
```

```
                    aux=abs(v-L0-u+K0)
                    aux2=(u-K0+v-L0-dux)/2
                    aux2[np.nonzero(aux2<0)]=0
                    val = val + coeff \
                            * Fraction(binomialCoefficient(d-1+j,j),
                                    factorial(2*d-1+j)\
                               * factorial(d-1-j))
                               * aux**(d-1-j) * aux2**(2*d-1+j)
    return val
```

Stiffness Matrix

Now we present the core function for the computation of the Stiffness Matrix: the algorithm for $\left(\frac{\partial^2 \chi_n}{(\partial r_1)^2} + \frac{\partial^2 \chi_n}{(\partial r_2)^2} + \frac{\partial^2 \chi_n}{(\partial r_1)(\partial r_2)}\right)(u_0 + n, v_0 + n)$.

```
def stiffness_matrix(u0,v0,n):
    """
        Computation of (\partial_{r_1, r_1} + \partial_{r_2, r_2}
        - \partial_{r_1, r_2}) \chi_{d,d,d} (u_0 + d, v_0 + d)
    """
    u = np.copy(u0)
    v = np.copy(v0)
    x = u + v
    sqrt3y = v - u
    x = -dbs(x)
    sqrt3y = abs(sqrt3y)
    u = (x - sqrt3y)/2
    v = (x + sqrt3y)/2

    id=np.nonzero(v>0)
    v[id]=-v[id]
    u[id]=u[id]+v[id]

    id=np.nonzero(v>u/2)
    v[id]=u[id]-v[id]
    u = u.astype(int)
    v = v.astype(int)

    val=np.zeros(u.shape, dtype=Fraction)

    for K in range(-d, int(ceil(np.max(u)))):
```

```
        K0=Fraction(K)
        for L in range(-d, int(ceil(np.max(v)))):
            L0=Fraction(L)
            for i in range(0, min(n+K, d+L)+1):
           coeff=Fraction((-1)**(K+L+i))*binomialCoefficient(d,i-K)\
                *binomialCoefficient(d,i-L)*binomialCoefficient(d,i)
                for j in range(0, n):
                    aux=abs(v-L0-u+K0)
                    aux2=(u-K0+v-L0-dux)/2
                    aux2[np.nonzero(aux2<0)]=Fraction(0)
                  val = val + Fraction(binomialCoefficient(n-1+j,j),
                              factorial(2*n-3+j)*factorial(n-1-j))
                           * coeff * aux**(n-1-j) * aux2**(2*n-3+j)
                    if n-1-j > 0:
                        val = val -Fraction(3) * coeff \
                            * Fraction(binomialCoefficient(n-1+j,j),
                               factorial(2*n-2+j)*factorial(n-2-j))\
                                 * aux**(n-2-j) * aux2**(2*n-2+j)
                    #end if
                    if n-2-j > 0:
                        val = val + Fraction(3) * coeff \
                            * Fraction(binomialCoefficient(n-1+j,j),
                              factorial(2*n-1+j)*factorial(n-3-j)) \
                                  * aux**(n-3-j) * aux2**(2*n-1+j)

    return val
```

Matrices construction

This last function uses the two algorithms previously defined to construct the matrices (minus a coefficient, see the following tables).

```
def mass_stiff_matrices(n0):
    """
     Computation of the coefficients which will be found in the mass
     and stiffness matrices with Box-splines of degree n0.
     The result is two matrices (let's call them MM and SM).

     For a Box-spline centered in (k_{1,i},k_{2,i}) and another one
     centered in (k_{1,j},k_{2,j}), the corresponding coefficient for
     the mass matrix (resp. for the stiffness matrix) at position
```

```
  (i,j) is the coefficient of MM (resp. SM) at position:
    (2*n0 + (k_{1,i} - k_{1,j})/h, 2*n0 + (k_{2,i} - k_{2,j})/h)
"""
n = 2*n0
u = range(-n+1, n)
v = range(-n+1, n)
U, V = np.meshgrid(u,v)
U = np.array(np.round(U), dtype=Fraction)
V = np.array(np.round(V), dtype=Fraction)
print(U)
print(V)

mM = mass_matrix(U,V,n)
sM = stiffness_matrix(U,V,n)

return sp.Matrix(mM), sp.Matrix(- sM)
```

Results

We list in this section the results obtained for different degrees. Let us define two Box-splines, the first one centered on the point of global index i, *i.e.* of hexagonal coordinates $(k_{1,i}, k_{2,i})$, and the second one centered on the point of global index j, or $(k_{1,j}, k_{2,j})$. Both splines are on a hexagonal mesh of step $h = L/N_c$, and are of degree (d_1, d_2, d_3) in each direction. Thus, to find the value of the Mass and Stiffness value, we see the Table corresponding to the degree (d_1, d_2, d_3), and the value at $(d_1 + d_3 + \frac{k_{1,i}-k_{1,j}}{h}, d_2 + d_3 + \frac{k_{2,i}-k_{2,j}}{h})$. Furthermore, the results for the Mass Matrix need to be multiplied by $\frac{2h^2}{\sqrt{3}}$ and by $\frac{2}{\sqrt{3}}$ for the Stiffness Matrix.

Mass matrix coefficients $\div \frac{2L^2}{N_c^2\sqrt{3}}$

c_1 \ c_2	1	2	3
1	$\frac{1}{12}$	$\frac{1}{12}$	0
2	$\frac{1}{12}$	$\frac{1}{2}$	$\frac{1}{12}$
3	0	$\frac{1}{12}$	$\frac{1}{12}$

Stiffness matrix coefficients $\div \frac{2}{\sqrt{3}}$

c_1 \ c_2	1	2	3
1	0.5	0.5	0
2	0.5	5	0.5
3	0	0.5	0.5

Table A.1: Non-null terms of the Mass and Stiffness matrices for box-splines of degree 1

Mass matrix coefficients $\div 181440 \frac{2L^2}{N_c^2\sqrt{3}}$

c_1 \ c_2	1	2	3	4	5	6	7
1	1	17	17	1	0	0	0
2	17	868	2550	868	17	0	0
3	17	2550	18871	18871	2550	17	0
4	1	868	18871	47496	18871	868	1
5	0	17	2550	18871	18871	2550	17
6	0	0	17	868	2550	868	17
7	0	0	0	1	17	17	1

Table A.2: Non-null terms of the Mass matrix for box-splines of degree 2

Stiffness matrix coefficients $\div 20160 \frac{2}{\sqrt{3}}$

c_1 \ c_2	1	2	3	4	5	6	7
1	-1	-43	-43	-1	0	0	0
2	-43	108	-1218	108	-43	0	0
3	-43	-1218	8153	8153	-1218	-43	0
4	-1	108	8153	23800	8153	108	-1
5	0	-43	-1218	8153	8153	-1218	-43
6	0	0	-43	108	-1218	108	-43
7	0	0	0	-1	-43	-43	-1

Table A.3: Non-null terms of the Stiffness matrix for box-splines of degree 2

Mass matrix coefficients $\div 83026944000 \frac{2L^2}{N_c^2\sqrt{3}}$

c_1 \ c_2	1	2	3	4	5	6	7	8	9	10	11
1	1	137	902	902	137	1	0	0	0	0	0
2	137	63850	962567	2181772	962567	63850	137	0	0	0	0
3	902	962567	28731267	129047744	129047744	28731267	962567	902	0	0	0
4	902	2181772	129047744	1091379764	2138034676	1091379764	129047744	2181772	902	0	0
5	137	962567	129047744	2138034676	7859593436	7859593436	2138034676	129047744	962567	137	0
6	1	63850	28731267	1091379764	7859593436	14746899204	7859593436	1091379764	28731267	63850	1
7	0	137	962567	129047744	2138034676	7859593436	7859593436	2138034676	129047744	962567	137
8	0	0	902	2181772	129047744	1091379764	2138034676	1091379764	129047744	2181772	902
9	0	0	0	902	962567	28731267	129047744	129047744	28731267	962567	902
10	0	0	0	0	137	63850	962567	2181772	962567	63850	137
11	0	0	0	0	0	1	137	902	902	137	1

Table A.4: Non-null terms of the Mass matrix for Box-splines of degree 3

Mass matrix coefficients $\div 207567360 \frac{2}{\sqrt{3}}$

c_1 \ c_2	1	2	3	4	5	6	7	8	9	10	11
1	-1	-197	-1993	-1993	-197	-1	0	0	0	0	0
2	-197	-13978	-267070	-1285700	-267070	-13978	-197	0	0	0	0
3	-1993	-267070	-842859	-10398707	-10398707	-842859	-267070	-1993	0	0	0
4	-1993	-1285700	-10398707	44735820	-28203916	44735820	-10398707	-1285700	-1993	0	0
5	-197	-267070	-10398707	-28203916	548369948	548369948	-28203916	-10398707	-267070	-197	0
6	-1	-13978	-842859	44735820	548369948	1147752060	548369948	44735820	-842859	-13978	-1
7	0	-197	-267070	-10398707	-28203916	548369948	548369948	-28203916	-10398707	-267070	-197
8	0	0	-1993	-1285700	-10398707	44735820	-28203916	44735820	-10398707	-1285700	-1993
9	0	0	0	-1993	-267070	-842859	-10398707	-10398707	-842859	-267070	-1993
10	0	0	0	0	-197	-13978	-267070	-1285700	-267070	-13978	-197
11	0	0	0	0	0	-1	-197	-1993	-1993	-197	-1

Table A.5: Non-null terms of the Stiffness matrix for Box-splines of degree 3

h-refinement with Box-splines elements

In the Finite Element method context, we refer to h-refinement for the procedure of dividing the elements into smaller ones. It is frequently associated to p-refinements, which refer to increasing the degree of the basis used for the definition of the FEM.

Analytical computation

The objective of this section is to find a relationship between χ_Ξ and $\chi_\Xi(2\mathbf{v} - \mathbf{v_1})$ with $\mathbf{v_1} \in \mathbb{Z}^2$. Therefore, we study the Fourier transform of χ_Ξ. In fact, this transform can be easily written by using Equation (A.4),

$$\begin{aligned}
\widehat{\chi_\Xi}(\zeta) &= \int \chi_\Xi(v)\, e^{-i\zeta\cdot\mathbf{v}} d\mathbf{v} \\
&= \int_{[0,1]^n} \exp\left(-i\,\zeta\cdot\sum_{\mathbf{v_i}\in\Xi} t_i\,\mathbf{v_i}\right) dt_1 ... dt_n \\
&= \int_{[0,1]^n} \prod_{\mathbf{v_i}\in\Xi} \exp\left(-i\,\zeta\cdot t_i\,\mathbf{v_i}\right) dt_1 ... dt_n \\
&= \prod_{v_i\in\Xi} \int_{[0,1]} \exp\left(-i\,t_i\,\zeta\cdot v_i\right) dt_i \\
&= \prod_{v_i\in\Xi} \frac{1 - e^{-i\zeta\cdot\mathbf{v_i}}}{i\,\zeta\cdot\mathbf{v_i}}.
\end{aligned} \tag{A.11}$$

Remark 10. *We notice that this expressions show that the order of the vectors in generating matrix is not important.*

Let $\chi_{\Xi,2} : \mathbf{v} \mapsto \chi_\Xi(2\mathbf{v})$, it yields

$$\begin{aligned}
\widehat{\chi_{\Xi,2}}(\zeta) &= \frac{1}{4}\,\widehat{\chi_\Xi}(\frac{\zeta}{2}) \\
&= 2^{n-2} \prod_{\mathbf{v_i}\in\Xi} \frac{1 - e^{-i\frac{\zeta}{2}\cdot\mathbf{v_i}}}{i\,\zeta\cdot\mathbf{v_i}}.
\end{aligned} \tag{A.12}$$

We divide Equation (A.11) by (A.12), we obtain

$$\frac{\widehat{\chi_\Xi}(\zeta)}{\widehat{\chi_{\Xi,2}}(\zeta)} = 2^{2-n} \prod_{v_i\in\Xi} \left(1 + e^{-i\frac{\zeta}{2}\cdot\mathbf{v_i}}\right)$$

□

Let see the results for each type of Box-splines we have seen.

Type-I Box-splines of constant degree on each direction

Let us set the generating matrix $\Xi = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ with multiplicity d for each vector. We get

$$\begin{aligned}
\frac{\widehat{\chi_\Xi}(\zeta)}{\widehat{\chi_{\Xi,2}}(\zeta)} &= 2^{2-3d} \left(1 + e^{-i\frac{\zeta}{2}\cdot\mathbf{v}_1}\right)^d \left(1 + e^{-i\frac{\zeta}{2}\cdot\mathbf{v}_2}\right)^d \left(1 + e^{-i\frac{\zeta}{2}\cdot\mathbf{v}_3}\right)^d \\
\frac{\widehat{\chi_\Xi}(2\zeta)}{\widehat{\chi_{\Xi,2}}(2\zeta)} &= 2^{2-3d} \left(\sum_{j=0}^{d} \binom{d}{j} e^{-i\,j\,\zeta\cdot\mathbf{v}_1}\right) \left(\sum_{k=0}^{d} \binom{d}{k} e^{-i\,k\,\zeta\cdot\mathbf{v}_2}\right) \left(\sum_{l=0}^{d} \binom{d}{l} e^{-i\,l\,\zeta\cdot\mathbf{v}_3}\right) \\
&= 2^{2-3d} \left(\sum_{j,k,l=0}^{d} \binom{d}{j}\binom{d}{k}\binom{d}{l} e^{-i\,\zeta\cdot((j+l)\mathbf{v}_1+(k+l)\mathbf{v}_2)}\right) \\
&= 2^{2-3d} \left(\sum_{\gamma,\lambda=0}^{2d} \sum_{l=\max(\gamma,\lambda,d)-d}^{\min(\gamma,\lambda,d)} \binom{d}{\gamma-l}\binom{d}{\lambda-l}\binom{d}{l} e^{-i\,\zeta\cdot(\gamma\mathbf{v}_1+\lambda\mathbf{v}_2)}\right) \\
&= 2^{2-3d} \left(\sum_{\gamma,\lambda=0}^{2d} \sum_{l=0}^{d} \binom{d}{\gamma-l}\binom{d}{\lambda-l}\binom{d}{l} e^{-i\,\zeta\cdot(\gamma\mathbf{v}_1+\lambda\mathbf{v}_2)}\right)
\end{aligned}$$

$$\widehat{\chi_\Xi}(\zeta) = 2^{2-3d} \left(\sum_{\gamma,\lambda=0}^{2d} \sum_{l=0}^{d} \binom{d}{\gamma-l}\binom{d}{\lambda-l}\binom{d}{l} e^{-i\,\frac{\zeta}{2}\cdot(\gamma\mathbf{v}_1+\lambda\mathbf{v}_2)}\right) \widehat{\chi_{\Xi,2}}(\zeta). \quad \text{(A.13)}$$

It yields

$$\chi_\Xi = 2^{2-3d} \chi_{\Xi,2} * b,$$

where

$$b = \sum_{\gamma,\lambda=0}^{2d} \left(\sum_{l=0}^{d} \binom{d}{\gamma-l}\binom{d}{\lambda-l}\binom{d}{l}\right) \delta_{\frac{\gamma\mathbf{v}_1+\lambda\mathbf{v}_2}{2}}.$$

The coefficients $\delta_{\mathbf{v}}$ are Dirac functions. It follows

$$\chi_\Xi(\mathbf{v}) = 2^{2-3d} \sum_{\gamma,\lambda=0}^{2d} \left(\sum_{l=0}^{d} \binom{d}{\gamma-l}\binom{d}{\lambda-l}\binom{d}{l}\right) \chi_\Xi(2\mathbf{v} - \gamma\mathbf{v}_1 - \lambda\mathbf{v}_2). \quad \text{(A.14)}$$

We compute the values of $u_{\gamma,\lambda,d} = \sum_{l=0}^{d} \binom{d}{\gamma-l}\binom{d}{\lambda-l}\binom{d}{l}$ for $d = 1, \ldots, 3$ with the following Equation

$$\begin{aligned}
u_{\gamma,\lambda,d+1} = 2\,u_{\gamma-1,\lambda-1,d} \;+\; u_{\gamma,\lambda,d} + u_{\gamma,\lambda-1,d} + u_{\gamma-1,\lambda,d} \\
+ u_{\gamma-2,\lambda-1,d} + u_{\gamma-1,\lambda-2,d} + u_{\gamma-2,\lambda-2,d}
\end{aligned} \quad \text{(A.15)}$$

Remark 11. *We notice that, for $d = 0$, the only non-null value is at the origin and is equal to* 1.

Using Equation (A.15), we get the values of $u_{\gamma,\lambda,d}$ for $d = 1, \ldots, 3$ presented in the next tables.

γ \ λ	0	1	2
0	1	1	0
1	1	2	1
2	0	1	1

γ \ λ	0	1	2	3	4
0	1	2	1	0	0
1	2	6	6	2	0
2	1	6	10	6	1
3	0	2	6	6	2
4	0	0	1	2	1

Table A.6: Values of $u_{\gamma,\lambda,d}$ for $d = 1$ and $d = 2$

γ \ λ	0	1	2	3	4	5	6
0	1	3	3	1	0	0	0
1	3	12	18	12	3	0	0
2	3	18	39	39	18	3	0
3	1	12	39	56	39	12	1
4	0	3	18	39	39	18	3
5	0	0	3	12	18	12	3
6	0	0	0	1	3	3	1

Table A.7: Values of $u_{\gamma,\lambda,d}$ for $d = 3$

Type-I Box-splines of variable degree

Let us set the generating matrix $\Xi = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ with multiplicity d_1 for vector $\mathbf{v}_1$, d_2 for vector $\mathbf{v}_2$ and d_3 for vector $\mathbf{v}_3 = \mathbf{v}_1 + \mathbf{v}_2$. Using the same procedures, we get

$$\chi_\Xi(\mathbf{v}) = 2^{2-d_1-d_2-d_3} \sum_{\gamma=0}^{d_1+d_3} \sum_{\lambda=0}^{d_2+d_3} \left(\sum_{l=0}^{d_3} \binom{d_1}{\gamma-l} \binom{d_2}{\lambda-l} \binom{d_3}{l} \right) \chi_\Xi(2\mathbf{v} - \gamma\mathbf{v}_1 - \lambda\mathbf{v}_2) \tag{A.16}$$

Box-spline of 2 directions matrices and with arbitrary degree

Let us set the generating matrix $\Xi = (\mathbf{v}_1, \mathbf{v}_2)$ with multiplicity d_1 for vector $\mathbf{v}_1$ and d_2 for vector $\mathbf{v}_2$.

$$\begin{aligned}
\frac{\widehat{\chi_\Xi}(\zeta)}{\widehat{\chi_{\Xi,2}}(\zeta)} &= 2^{2-d_1-d_2} \left(1 + e^{-i\frac{\zeta}{2}\cdot\mathbf{v}_1}\right)^{d_1} \left(1 + e^{-i\frac{\zeta}{2}\cdot\mathbf{v}_2}\right)^{d_2} \frac{\widehat{\chi_\Xi}(2\zeta)}{\widehat{\chi_{\Xi,2}}(2\zeta)} \\
&= 2^{2-d_1-d_2} \left(\sum_{j=0}^{d_1} \binom{d_1}{j} e^{-i\,j\,\zeta\cdot\mathbf{v}_1} \right) \left(\sum_{k=0}^{d_2} \binom{d_2}{k} e^{-i\,k\,\zeta\cdot\mathbf{v}_2} \right) \\
&= 2^{2-d_1-d_2} \left(\sum_{j=0}^{d_1} \sum_{k=0}^{d_2} \binom{d_1}{j} \binom{d_2}{k} e^{-i\,\zeta\cdot(j\mathbf{v}_1 + k\mathbf{v}_2)} \right).
\end{aligned}$$

It yields

$$\chi_\Xi(\mathbf{v}) = 2^{2-d_1-d_2} \sum_{j=0}^{d_1} \sum_{k=0}^{d_2} \binom{d_1}{j} \binom{d_2}{k} \chi_\Xi(2\mathbf{v} - j\mathbf{v}_1 - k\mathbf{v}_2). \tag{A.17}$$

Type-II Box-splines of arbitrary degree

Let us set the generating matrix $\Xi = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4)$ with multiplicity d_1 for vector $\mathbf{v}_1$, d_2 for vector $\mathbf{v}_2$, d_3 for vector $\mathbf{v}_3$ and d_1 for the vector $\mathbf{v}_4 = \mathbf{v}_1 - \mathbf{v}_2$. We get

$$\begin{aligned}
\frac{\widehat{\chi_\Xi}(\zeta)}{\widehat{\chi_{\Xi,2}}(\zeta)} &= 2^{2-3d_1} \left(1 + e^{-i\frac{\zeta}{2}\cdot\mathbf{v}_1}\right)^{d_1} \left(1 + e^{-i\frac{\zeta}{2}\cdot\mathbf{v}_2}\right)^{d_2} \left(1 + e^{-i\frac{\zeta}{2}\cdot\mathbf{v}_3}\right)^{d_3} \\
&\qquad \left(1 + e^{-i\frac{\zeta}{2}\cdot\mathbf{v}_4}\right)^{d_1} \frac{\widehat{\chi_\Xi}(2\zeta)}{\widehat{\chi_{\Xi,2}}(2\zeta)} \\
&= 2^{2-d_1-d_2-d_3-d_1} \left(\sum_{j=0}^{d_1} \binom{d_1}{j} e^{-i\,j\,\zeta\cdot\mathbf{v}_1} \right) \left(\sum_{k=0}^{d_2} \binom{d_2}{k} e^{-i\,k\,\zeta\cdot\mathbf{v}_2} \right) \\
&\qquad \left(\sum_{l=0}^{d_3} \binom{d_3}{l} e^{-i\,l\,\zeta\cdot\mathbf{v}_3} \right) \left(\sum_{m=0}^{d_1} \binom{d_1}{m} e^{-i\,m\,\zeta\cdot\mathbf{v}_4} \right)
\end{aligned}$$

$$\frac{\widehat{\chi_\Xi}(\zeta)}{\widehat{\chi_{\Xi,2}}(\zeta)} = 2^{2-2d_1-d_2-d_3} \sum_{j=0}^{d_1}\sum_{k=0}^{d_2}\sum_{l=0}^{d_3}\sum_{m=0}^{d_1} \binom{d_1}{j}\binom{d_2}{k}\binom{d_3}{l}\binom{d_1}{m} e^{-i\,\zeta\cdot((j+l+m)\mathbf{v}_1+(k+l-m)\mathbf{v}_2)}$$
$$= 2^{2-2d_1-d_2-d_3} \sum_{\gamma=0}^{2d_1+d_3}\sum_{\lambda=-d_1}^{d_2+d_3}\sum_{l=0}^{d_3}\sum_{m=0}^{d_1} \binom{d_1}{\gamma-l-m}\binom{d_2}{\lambda-l+m}\binom{d_3}{l}\binom{d_2}{m} e^{-i\,\zeta\cdot(\gamma\mathbf{v}_1+\lambda\mathbf{v}_2)}.$$

Finally, we get

$$\chi_\Xi(\mathbf{v}) = 2^{2-2d_1-d_2-d_3} \sum_{\gamma=0}^{2d_1+d_3}\sum_{\lambda=-d_1}^{d_2+d_3} \left(\sum_{l=0}^{d_3}\sum_{m=0}^{d_1} \binom{d_1}{\gamma-l-m}\binom{d_2}{\lambda-l+m}\binom{d_3}{l}\binom{d_1}{m}\right) \chi_\Xi(2\mathbf{v}-\gamma\mathbf{v}_1-\lambda\mathbf{v}_2) \tag{A.18}$$

Quasi-interpolation and pre-filters

The equation defining the pre-filters is given by Condat and Van de Ville [CVDV07]

$$\frac{1}{\widehat{p}(\omega)} = \widehat{B_\Xi}(\omega) + \mathrm{O}(||\omega||^L)$$

We consider the Box-splines for the hexagonal mesh as defined in Section 2. They are of constant degree along each direction. Let n be that degree, thus $L = n$ in the formula above. Furthermore, the two pre-filters IIR1 and IIR2 that we will consider in this manuscript, are defined by their $\mathscr{Z}$-transform. The latter is defined for a discrete signal $(s[\mathbf{k}])_{\mathbf{k}\in\mathbb{Z}^2}$ by

$$S(\mathbf{z}) = \sum_{\mathbf{k}\in\mathbb{Z}^2} s[\mathbf{k}]\, z_1^{-k_1} z_2^{-k_2}.$$

The Fourier transform follows from

$$\widehat{p}(\omega) = P(e^{i\,\omega_1}, e^{i\,\omega_2}),$$

with $\omega_i = \omega \cdot u_i$.

The $\mathscr{Z}$-transforms of the two pre-filters (found in the article cited above) are

$$P_{IIR1}(\mathbf{z}) = \frac{1}{\sum_{k=0}^{K} h[k]\,\mathrm{ring}_k(\mathbf{z})} \tag{A.19}$$

$$P_{IIR2}(\mathbf{z}) = \frac{1}{H(z_1)H(z_2)H(z_1\,z_2^{-1})} \tag{A.20}$$

with ring_k, the $\mathscr{Z}$-transform of the signal (it is equal to 1 at the k-th ring and 0 elsewhere), and $H(z) = h[0] + \sum_{k=1}^{n} h[k]\,(z^k + z^{-k})$. The $h[k]$ coefficients are to be computed for each pre-filter.

Pre-filter IIR 2

Let us compute equation (A.20)

$$H(e^{i\,\omega_1})H(e^{i\,\omega_2})H(e^{i\,(\omega_1-\omega_2)}) = \left(\frac{1-e^{-i\omega\cdot\mathbf{u_1}}}{i\,\omega\cdot\mathbf{r}_1}\right)^n \left(\frac{1-e^{-i\omega\cdot\mathbf{u_2}}}{i\,\omega\cdot\mathbf{r}_2}\right)^n \left(\frac{1-e^{-i\omega\cdot\mathbf{u_3}}}{i\,\omega\cdot\mathbf{r}_3}\right)^n + \mathrm{O}(||\omega||^L).$$

Since $\mathbf{r}_3 = \mathbf{r}_1 + \mathbf{r}_2$, the equation can be simplified to

$$H(e^{i\,x}) = \left(\frac{1-e^{-ix}}{i\,x}\right)^n + \mathrm{O}(x^n).$$

By expanding this equation, we get n equations for each order. The following Python script creates the corresponding matrix and vector to this system, and solves the system by inverting the matrix.

```
import numpy as np
import fractions as fr
import math
import sympy as sp

def makematrix(n):
    """ Creation of the Matrix containing the RHS of the system
    to compute the h[k] coefficients for the piir2 pre-filter"""

    M=sp.zeros((n+1,n+1))
    for i in range(0,n+1):
        M[0,i]=fr.Fraction(2)
        for j in range(1,n+1):
          M[j,i]=(-1)**j*fr.Fraction(2*i**(2*j),math.factorial(2*j))
    M[0,0]=fr.Fraction(1)
return M
```

```
def makelambd(n):

    lambd=sp.zeros((n+1,1))
    j=np.zeros(n, dtype=int, order='C')
    while j[0]!=n:
        increase(j, n)
        s=sum(j)
        if s<=n:
            coeff=fr.Fraction(1)
            for i in range(0,n):
                coeff=coeff/fr.Fraction(math.factorial(2*j[i]+1))
            lambd[s]=lambd[s]+coeff

    for i in range(1,n+1):
        lambd[i]=lambd[i]*fr.Fraction(1, 4**i)*(-1)**i
    lambd[0]=fr.Fraction(1)

return lambd

def increase(j, n):
    if sum(j) == n:
        i=-1
        while j[i] == 0:
            i=i-1
        j[i]=0
        j[i-1]=j[i-1]+1
    else:
        j[-1]=j[-1]+1
return

def coeff_computation(n):
    """ Creation and solving of the system to compute
    the h[k] coefficients for the piir2 pre-filter

    n: degree of the box-splines"""

    M=makematrix(n)
    V=makelambd(n)
return M.inv()*V
```

The function `coeff_computation(n)` returns the coefficients $h[\mathbf{k}]$ wanted. The following table contains the results for the degree 1, 2 and 3.

Degree	1	2	3
$h[0]$	$\frac{11}{12}$	$\frac{97}{120}$	$\frac{173863}{241920}$
$h[1]$	$\frac{1}{24}$	$\frac{1}{10}$	$\frac{47309}{322560}$
$h[2]$		$-\frac{1}{240}$	$-\frac{209}{32256}$
$h[3]$			$\frac{457}{967680}$

Table A.8: $h[\mathbf{k}]$ coefficients for the PIIR2 pre-filter with Box-splines of degree $n = 1, 2, 3$

Pre-filter IIR 1

There is no straight-forward simplification for the computation of the pre-filter IIR 1. Thus, we created a symbolic code to solve Equation (A.19). We chose $\omega_2 = 0$. However, a verification at the end is needed, to test if the coefficients are correct for all ω_2.

```
from math import *
import sympy as sp
import numpy as np
from fractions import *
import matplotlib.pyplot as mpp
import cmath

def ringradiussquare(n_max):
    """
    Returns 2 results:
    radius is a dict which gives the 2-norm to the square of
    vectors n1 r_1 + n2 r_2 with n1 and n2 integers,
    such that M = n1+n2 and N = n1-n2. This value is then given by
    (N**2 + 3 * M**2)/4. We take points such that this value is
    less than n_max.
    Rradius [sorted list] contains all radius which are in 'radius'
    """
    interm=set()
    radius={}
    for N in range(-2*n_max, 2*n_max+1):
```

```
        a=int(math.sqrt((4*n_max**2-N**2)/3))
        mod = N % 2
        for x in range(-int((a+mod)/2), int((a-mod)/2)+1):
            M = mod + 2*x
            r = round((N**2+3*M**2)/4)
            interm = interm | set([r])
            radius[N,M] = r
    Rradius=sorted(list(interm))
return Rradius, radius

def ringnumber(Rradius, radius, K):
    """
        Return the circle number of all points of radius s.t.
        its circle number is less than K
    """
    Rnumber={}
    for ((N,M),r) in radius.items():
        numb = Rradius.index(r)
        if numb <= K:
            Rnumber[N,M] = numb
return Rnumber

def make_coeff_function(Rnumber, coeff, n):
    """
     Return the coefficients of the Taylor expansion of p(\omega_1, 0)
        depending on h[0], ..., h[n] given in coeff.
        coeff_function[i] corresponds to the coefficient of omega**i
    """
    coeff_function = sp.zeros(1, n+1)
    for (N,M) in Rnumber.keys():
        if (N,M) == (0,0):
            coeff_function[0] = coeff_function[0] + coeff[0]
        elif (N,M) > (0,0):
         # Each point is linked with its opposite in order to have a cos
            for i in range(0, n+1):
                coeff_function[i] = coeff_function[i] \
              + 2*(-1)**i*(Fraction((N+M),2))**(2*i)/factorial(2*i)\
                * coeff[Rnumber[N,M]]
return coeff_function
```

```
def make_function(Rnumber, coeff_function_modif, n_modif):
    """
        Build the function of the Taylor expansion of p(\omega_1, 0)
        with the coefficients coeff_function_modif
        until the order n_modif
    """
    global omega
    omega = sp.Symbol('omega')
    p = 0
    for i in range(0,n_modif+1):
        p = p + coeff_function_modif[i]*omega**(2*i)
return p


def makelambd_2(n):
    """
        Compute the coefficients of the Taylor expansion of
        B_n(\omega_1, 0) until the order n
    """
    lambd=sp.zeros((n+1,1))
    j=np.zeros(n, dtype=int, order='C')
    while j[0]!=n:
        increase(j, n)
        s=sum(j)
        if s<=n:
            coeff=Fraction(1)
            for i in range(0,n):
                coeff=coeff/Fraction(factorial(2*j[i]+1))
            lambd[s]=lambd[s]+coeff

    for i in range(1,n+1):
        lambd[i]=lambd[i]*Fraction(1, 4**i)*(-1)**i

    lambd[0]=Fraction(1)

return lambd


def coeff_computation_2(n):
    """
```

```
        Principle function which builds the functions,
        and the equations and solves them.
    """
    Rradius, radius = ringradiussquare(n)
    Rnumber = ringnumber(Rradius, radius, n)
    lambd = makelambd_2(2*n)

    # Plot of the taken points
    x = [N[1]/2 for N in Rnumber.keys()]
    y = [N[0]*sqrt(3)/2 for N in Rnumber.keys()]
    mpp.plot(x, y, 'o')

    # Definition of the symbolic coefficients h[i], in coeff
    coeff = {}
    for i in range(0,n+1):
        coeff[i] = sp.Symbol("".join([' h', str(i), ' ']))

# Build of the coefficients of the Taylor expansion of p(\omega_1,0)
    coeff_function = make_coeff_function(Rnumber, coeff, n)
    coeff_function_modif = coeff_function[:,:]

    # Matrix which countains the linear equations
    Lineq = sp.zeros(1, n+1)

    # Beginning of the loop which builds the equations
    for i in range(0,n+1):
     # Computation of the i^th coefficient in the Taylor expansion of
       # 1/p(\omega_1, 0) taking into account of the previous equations
        p = make_function(Rnumber, coeff_function_modif, i)
        b = 1/p
        result=sp.diff(b, omega, 2*i).subs(omega, 0)/factorial(2*i)

        # for the first iteration, the equation
        # is 1/(linear equation in h[0], ..., h[n]) = 1
        if i == 0:
            result = 1/result

        # Linear equation put into the matrix
        Lineq[i] = result-lambd[i]
```

```
        # Modification of the coefficients by taking into account
        # the new equation
        coeff_function_modif[i] = \
        coeff_function_modif[i].subs(coeff[1], \
                                     sp.solve(result-lambd[i], \
                                              coeff[1])[0])

    # Computation of the results with "solve"
    results_1 = sp.solve(Lineq, list(coeff.values()), dict=True)
    print(' Results : ', results_1)
    if results_1 == []:
        print('No solution')
    else:
        results = sp.zeros(1,n+1)
        for i in range(0,n+1):
            results[i] = results_1[0][coeff[i]]

        coeff_validation(n, Rnumber, results)

return results


def coeff_validation(n, Rnumber, results):
    """
      Print the Taylor expansion of B_n(\omega_1,0) - 1/p(\omega_1,0)
      at order 2*(n+1) with the computed coefficients to validate them
    """
    B = (sp.sin(omega/2)*2/omega)**(2*n)
    b = 0
    for (N,M) in Rnumber.keys():
        if (M,N) > (0,0):
          b = b + 2 * results[Rnumber[N,M]] * sp.cos(((M+N)/2)*omega)
        elif (M, N) == (0, 0):
            b = b + results[0]
    print((B-1/b).series(omega, 0, 2*(n+1)))
return
```

The results are given by `coeff_computation_2(n)`. The script seems to work perfectly for degrees $n = 1$ and $n = 2$, however, there is no solution for $n \geq 3$.

Quasi-interpolation algorithm

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 (see the sections above), and an algorithm for computing the Box-splines (see Section 2). In these algorithms `PreFilter(l)` are the coefficients found by the last two scripts.

Algorithm 2: Quasi-interpolation with Box-splines

Data: Domain Ω; function f known at 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`)`;
  Initialize K $=$ number of points in the vicinity of $\mathbf{x}_i$ (depends on pre-filter);
  Compute hexagonal coordinates: $\tilde{\mathbf{k}} = [\lfloor u \rfloor \ \lfloor v \rfloor]$ where $(u\ v)^T = \mathbf{R}^{-1}\mathbf{x}$;
  for $\mathbf{k}_\ell = 0$ **to** K -1 **do**
    `/* Treat points on enveloping rhomboid of radius = degree */`
    Compute hexagonal coordinates of point in vicinity of $\mathbf{x}$ using
    $\mathbf{k}_\ell \longrightarrow \hat{\mathbf{k}} = \tilde{\mathbf{k}} + \mathbf{k}_\ell$;
    if $\hat{\mathbf{k}} \in \Omega$ **then**
      Compute global index of $\hat{\mathbf{k}} \longrightarrow$ index;
      Get cartesian coordinates of point at index $\longrightarrow \hat{\mathbf{x}} = \mathbf{R}\hat{\mathbf{k}}$;
      result $=$ result $+$ coeffs(index) * `BoxSplineValue(`$\hat{\mathbf{x}}$, degree`)`
    end
  end
end

Algorithm 3: (`BoxSplineCoeff`) Computation of Box-spline coefficients

Data: sample values of $f(\mathbf{x}_i)$; degree, the degree of the splines.
Result: coeffs, Box-splines coefficients at each mesh point $\mathbf{x}_i$
Initialize K $=$ number of points in the vicinity of $\mathbf{x}_i$ (depends on pre-filter);
Initialize PreFilter $=$ array of local pre-filters. ; `/* See Section (2) */`
forall the $\mathbf{k}_i \in \Omega$ **do**
  Initialize coeffs(i) $= 0$;
  for $\ell = 0$ **to** K -1 **do**
    Compute global index of point at $(\mathbf{k}_i + \mathbf{k}_\ell) \longrightarrow$ index;
    coeffs(i) $=$ coeffs(i) $+$ sample(index) * PreFilter(ℓ)
  end
end

B

Hexagonal mesh implementations

A big challenge during this thesis was to find the appropriate library for the implementations needed. We always looked for the best compromise between the workload and the benefits. The results, mostly for the hexagonal-mesh implementations, was somewhat confusing, so the following list should clarify any doubts.

Development in SeLaLib

The first development of the box-splines were done in SeLaLib. At the beginning, we were looking only to test the interpolation methods using box-splines. All tools and objects needed for the CEMRACS 2014 project were also developed in this library. In detail, the following list details what is included.

Hex-mesh (object) An object containing all the information relative to a hex-mesh. Functions to compute the cartesian and hexagonal coordinates. The numbering of points, cells, edges. Transformations (hexagonal mesh to circle, to ovals, to miller equilibriums, ...).

Box-splines (object) Computation of arbitrary degree splines. Optimized algorithm for box-splines of order 2. Derivatives.

Fekete quadrature points (functionality) Tools needed to create and use the Fekete quadrature points of arbitrary degree on an element of the hex-mesh.

Hex-pre-filters (functionality) Computation of different pre-filters for quasi-interpolations using box-splines (implementation of G. Ferriere algorithms).

Box-splines interpolation (functionality) Functions to compute the interpolation of a function on a hex-mesh.

Poisson solver (functionality) Finite differences solver of order 4 for the poisson equation. Assembly of stiffness matrix and rhs. Linear solver of system.

ODE solver (functionality) Euler and Adam solver to compute characteristics on a hex-mesh (C. Prouveur)

Guiding-center (simulation) Simulation of a guiding center model (advection + poisson) on a hex-mesh. Only on logical space.

(old) Django-interface (functionality) Functions that output the files needed by the gforge version of Django. It exports information about a hex-mesh, box-splines, connectivity, numbering and so on.

Development in Django

The developments in this library were made in order to develop a Finite Element elliptic solver on the hex-mesh. A second objective was the capability to treat the boundary conditions using the Nitsche method. Finally, Django was supposed to be used only as a external solver that would be coupled to the interpolation in SeLaLib. The list below contains the elements available in the gforge version of Django.

Matrix assembly (functionality) Procedure to assemble a 2D matrix of a triangular mesh and in particular for a hex-mesh.

Quadrature (object) Creates quadrature rules from the files imported from SeLaLib

Box-splines (object) Creates basis for the hex-mesh from the files imported from SeLaLib

Diffusion (simulation) Resolution of the diffusion on a hexagon, tested with BC=0.

Poisson (simulation) Resolution of the poisson equation on a hexagon, tested with BC=0

Nitsche (functionality) Added terms for treating the BC using the Nitsche method. Added directly to each model.

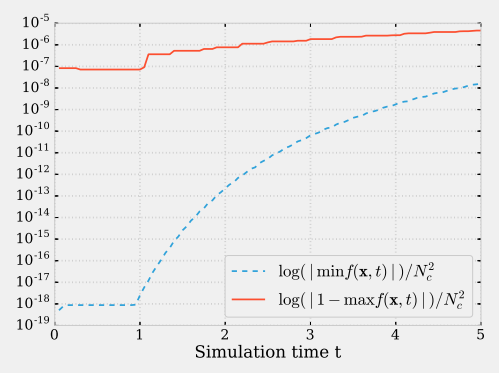
C

Results for different quasi-interpolation pre-filters

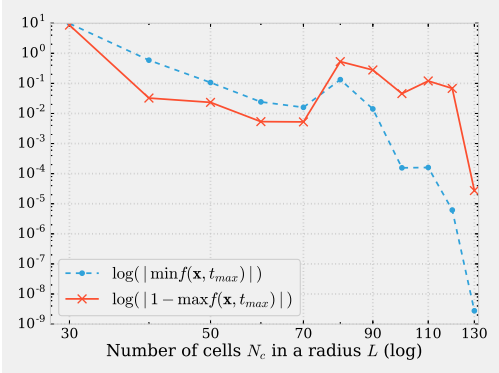
In this Appendix, we show the results of the Test-case 13 with different degrees and pre-filters. We recall the details of the test-case in the following paragraph.

> We present the results found for the constant advection equation Equation (4.1). Here, we try to recreate as close as possible Test-case 5. Thus, we set a linear advection model on hex-mesh of radius $L = 1$, centered at the origin. Since the radius of the mesh is $L = 1$, and the number of cells N_c corresponds to the number of cells in a radius, the mesh step $\Delta x = 1/N_c$ is equivalent to the mesh step in the Multi-patch Approach. Indeed, the interpolations for the MPSL scheme are done in the logical domain of length 1 (independently of the physical domain) and with a mesh refinement of N. Thus, to have comparable mesh steps we set N_c to N. We advect the same distribution function as in Test-case 5: a gaussian centered in $\mathbf{x}_c = (-0.4, -0.4)$ of variance $\sigma = 0.04$ and amplitude 1. The advection coefficient is $\mathbf{A} = (0.15, 0.15)$, while the parameters of the simulations are $t_{max} = 5$ with a time step $\Delta t = 0.05$. Finally, for the quasi-interpolation scheme, we used the pre-filter p_{FIR} based on the results of the previous test-case, and Box-splines of degree 1 and 2.

Test-case 17: Pre-filter p_{FIR} with degree 1 Box-splines

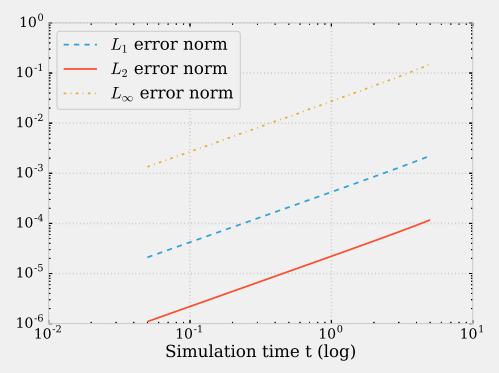


(a) Distribution bounds time evolution with $N_c = 100$

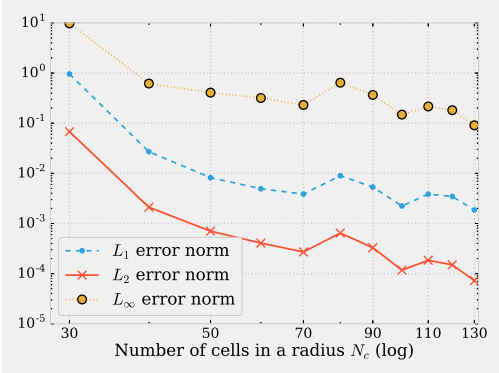


(b) Distribution bounds refinement evolution at $t_{max} = 5$

Figure C.1: Test-case 17: Bounds of the distribution function over time (left) with $N_c = 100$ and number of cells (right) at $t = 5$ with degree 1 Box-splines and p_{FIR} filter



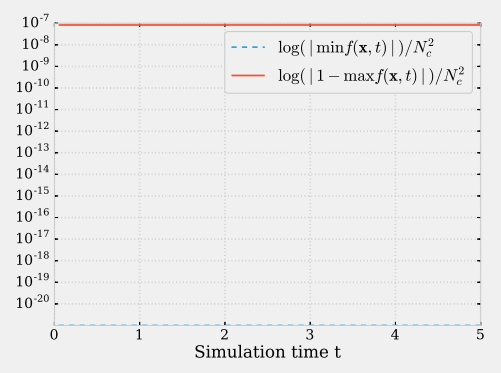
(a) Errors over time with $N_c = 100$



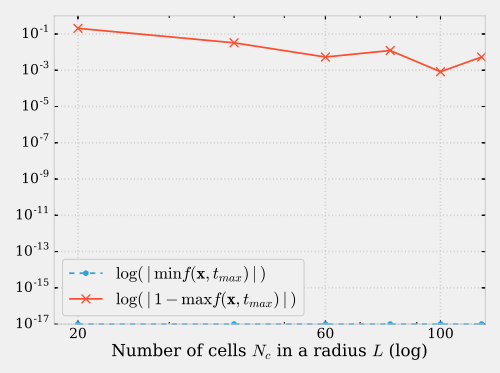
(b) Errors over mesh step at $t_{max} = 5$

Figure C.2: Test-case 17: Evolution of L_1, L_2 and L_∞ errors over time (left) with $N_c = 100$ and over number of cells (right) at $t = 5$ with degree 1 Box-splines and p_{FIR} filter

Test-case 18: Pre-filter p_{int} with degree 1 Box-splines

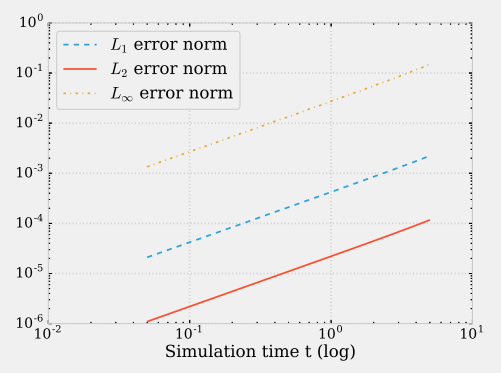


(a) Distribution bounds time evolution with $N_c = 100$

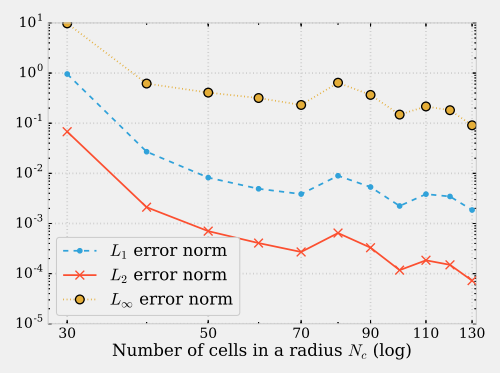


(b) Distribution bounds refinement evolution at $t_{max} = 5$

Figure C.3: Test-case 18: Bounds of the distribution function over time (left) with $N_c = 100$ and number of cells (right) at $t = 5$ with degree 1 Box-splines and p_{int} filter



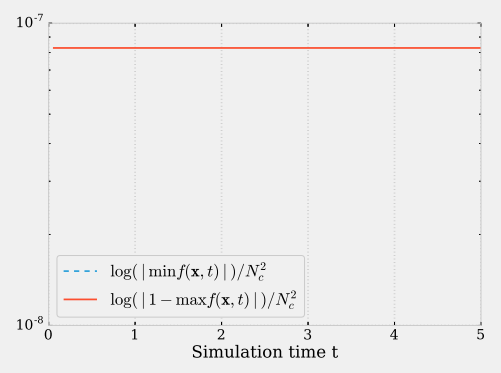
(a) Errors over time with $N_c = 100$



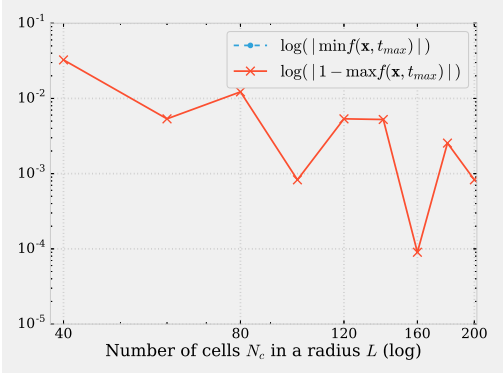
(b) Errors over mesh step at $t_{max} = 5$

Figure C.4: Test-case 18: Evolution of L_1, L_2 and L_∞ errors over time (left) with $N_c = 100$ and over number of cells (right) at $t = 5$ with degree 1 Box-splines and p_{int} filter

Test-case 19: Pre-filter p_{int} with degree 2 Box-splines

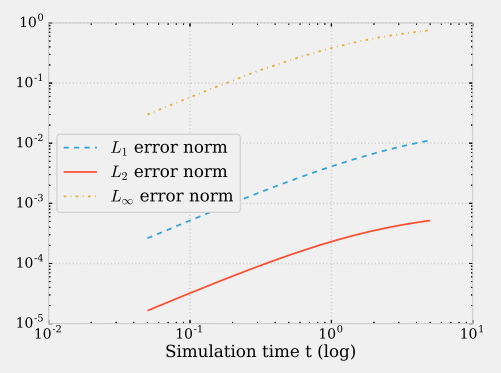


(a) Distribution bounds time evolution with $N_c = 100$

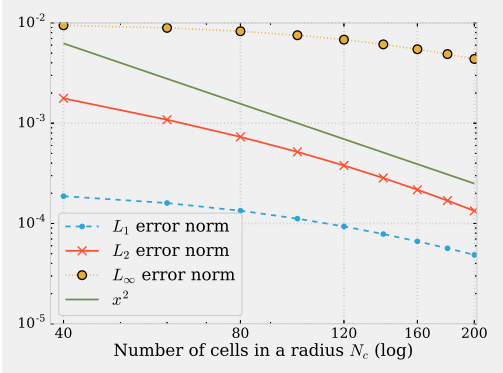


(b) Distribution bounds refinement evolution at $t_{max} = 5$

Figure C.5: Test-case 19: Bounds of the distribution function over time (left) with $N_c = 100$ and number of cells (right) at $t = 5$ with degree 2 Box-splines and p_{int} filter



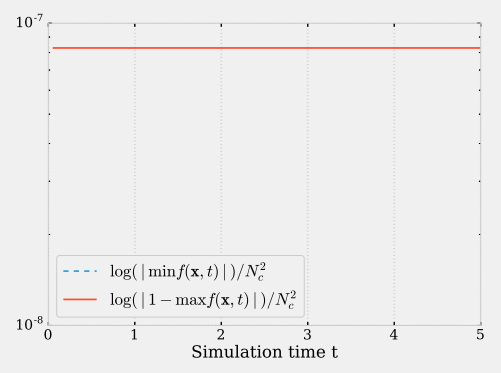
(a) Errors over time with $N_c = 100$



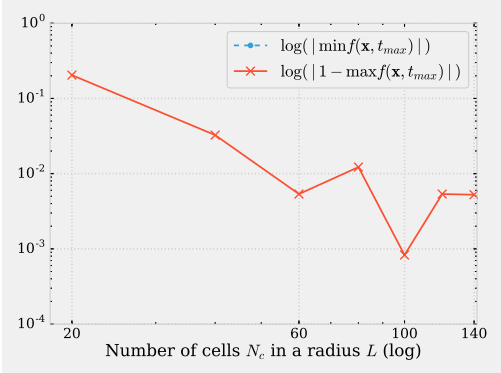
(b) Errors over mesh step at $t_{max} = 5$

Figure C.6: Test-case 19: Evolution of L_1, L_2 and L_∞ errors over time (left) with $N_c = 100$ and over number of cells (right) at $t = 5$ with degree 2 Box-splines and p_{int} filter

Test-case 20: Pre-filter p_{iir1} with degree 1 Box-splines

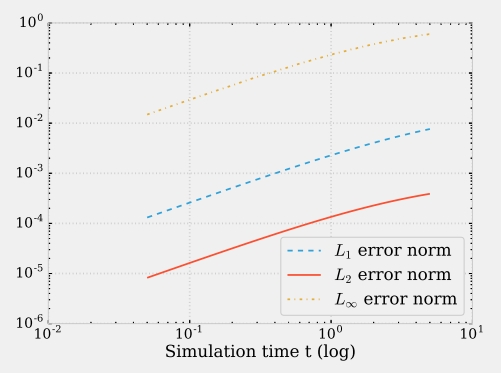


(a) Distribution bounds time evolution with $N_c = 100$

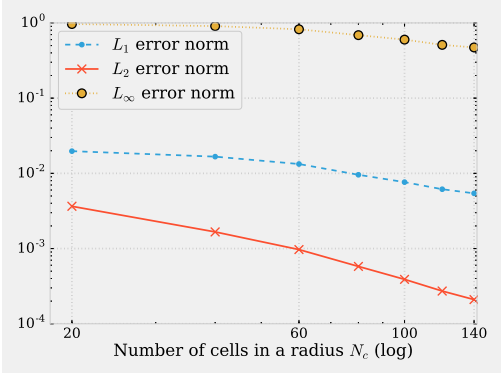


(b) Distribution bounds refinement evolution at $t_{max} = 5$

Figure C.7: Test-case 20: Bounds of the distribution function over time (left) with $N_c = 100$ and number of cells (right) at $t = 5$ with degree 1 Box-splines and p_{iir1} filter



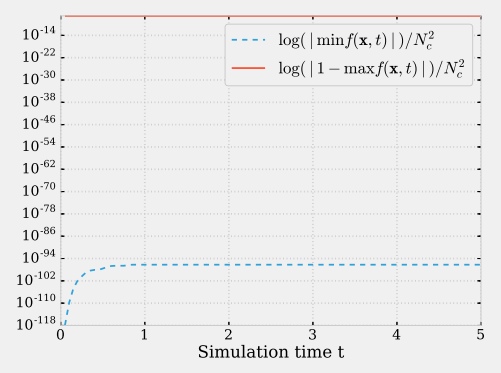
(a) Errors over time with $N_c = 100$



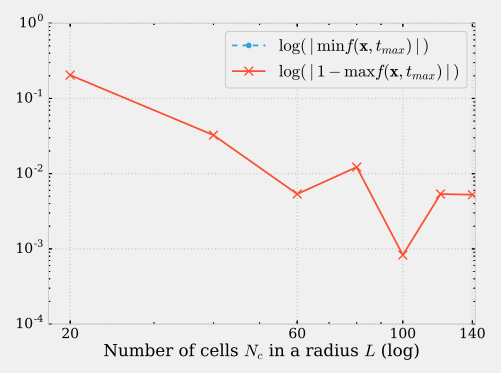
(b) Errors over mesh step at $t_{max} = 5$

Figure C.8: Test-case 20: Evolution of L_1, L_2 and L_∞ errors over time (left) with $N_c = 100$ and over number of cells (right) at $t = 5$ with degree 1 Box-splines and p_{iir1} filter

Test-case 21: Pre-filter p_{iir1} with degree 2 Box-splines

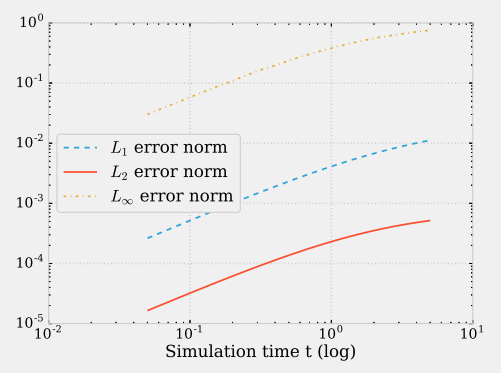


(a) Distribution bounds time evolution with $N_c = 100$

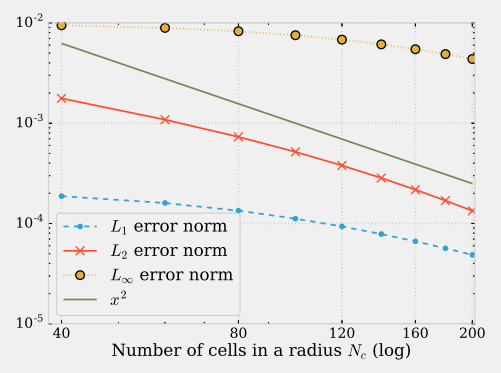


(b) Distribution bounds refinement evolution at $t_{max} = 5$

Figure C.9: Test-case 21: Bounds of the distribution function over time (left) with $N_c = 100$ and number of cells (right) at $t = 5$ with degree 2 Box-splines and p_{iir1} filter



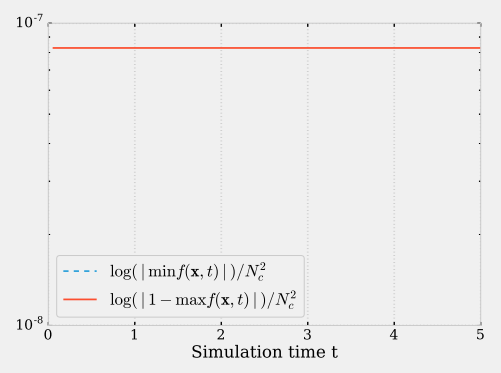
(a) Errors over time with $N_c = 100$



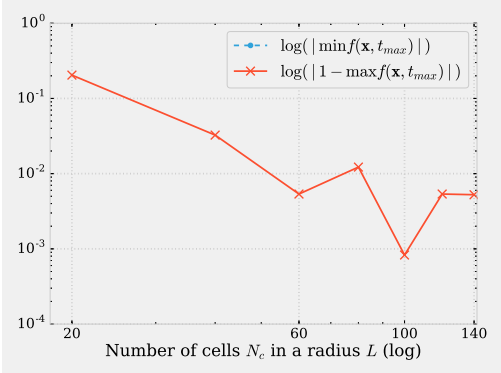
(b) Errors over mesh step at $t_{max} = 5$

Figure C.10: Test-case 21: Evolution of L_1, L_2 and L_∞ errors over time (left) with $N_c = 100$ and over number of cells (right) at $t = 5$ with degree 2 Box-splines and p_{iir1} filter

Test-case 22: Pre-filter p_{iir2} with degree 1 Box-splines

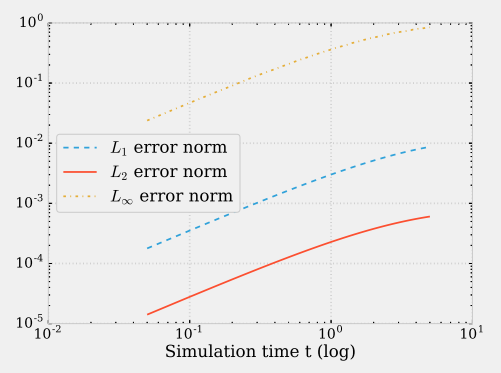


(a) Distribution bounds time evolution with $N_c = 100$

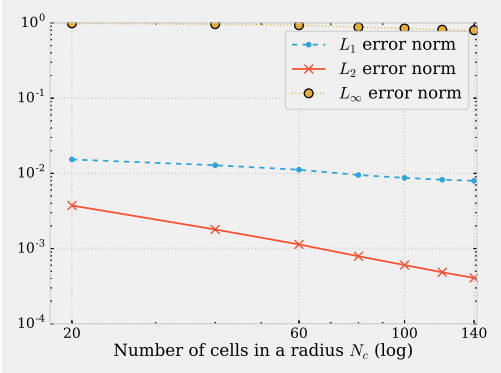


(b) Distribution bounds refinement evolution at $t_{max} = 5$

Figure C.11: Test-case 22: Bounds of the distribution function over time (left) with $N_c = 100$ and number of cells (right) at $t = 5$ with degree 1 Box-splines and p_{iir2} filter



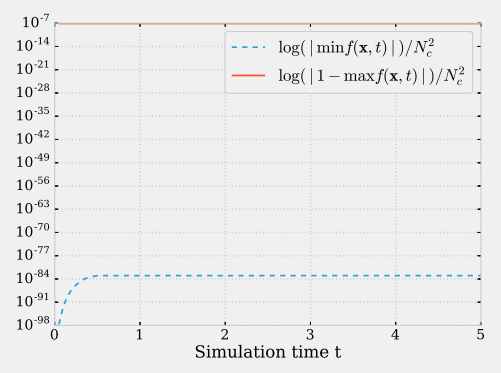
(a) Errors over time with $N_c = 100$



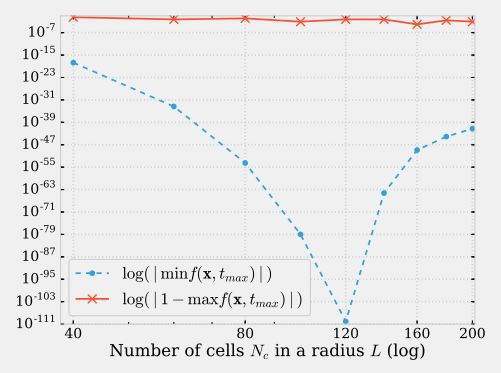
(b) Errors over mesh step at $t_{max} = 5$

Figure C.12: Test-case 22: Evolution of L_1, L_2 and L_∞ errors over time (left) with $N_c = 100$ and over number of cells (right) at $t = 5$ with degree 1 Box-splines and p_{iir2} filter

Test-case 23: Pre-filter p_{iir2} with degree 2 Box-splines

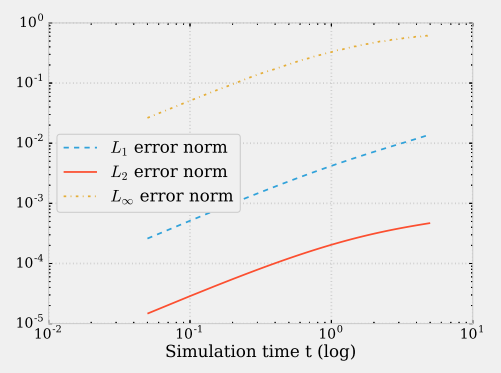


(a) Distribution bounds time evolution with $N_c = 100$

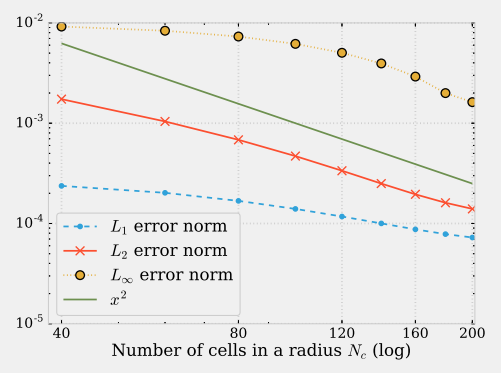


(b) Distribution bounds refinement evolution at $t_{max} = 5$

Figure C.13: Test-case 23: Bounds of the distribution function over time (left) with $N_c = 100$ and number of cells (right) at $t = 5$ with degree 2 Box-splines and p_{iir2} filter



(a) Errors over time with $N_c = 100$



(b) Errors over mesh step at $t_{max} = 5$

Figure C.14: Test-case 23: Evolution of L_1, L_2 and L_∞ errors over time (left) with $N_c = 100$ and over number of cells (right) at $t = 5$ with degree 2 Box-splines and p_{iir2} filter

Bibliography

[AGV+09] 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.

[ALG+11] 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.

[Ber94] M. Bernadou. *Methode d'elements finis pour les problemes de coques minces*. Masson editions, 1994.

[BH07] A. Brizard and T. Hahm. Foundations of nonlinear gyrokinetic theory. *Rev. Mod. Phys.*, 79(2):421–468, Apr 2007.

[BH10] 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.

[BHS13] B. D. Bojanov, H. Hakopian, and B. Sahakian. *Spline functions and multivariate interpolations*, volume 248. Springer Science & Business Media, 2013.

[BL85] C. K. Birdsall and A. B. Langdon. *Plasma Physics Via Computer*. McGraw-Hill, Inc., New York, NY, USA, 1985.

[BM08] N. Besse and M. Mehrenberger. Convergence of classes of high-order semi-lagrangian schemes for the vlasov-poisson system. *Mathematics of Computation*, 77(61):93–123, 2008.

[BS03] 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.

[BU99] T. Blu and M. Unser. Quantitative fourier analysis of approximation techniques. i. interpolators and projectors. *Signal Processing, IEEE Transactions on*, 47(10):2783–2795, 1999.

[CGH+14] 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.

[CHB09] J. A. Cottrell, T. J. Hughes, and Y. Bazilevs. *Isogeometric analysis: toward integration of CAD and FEA*. John Wiley & Sons, 2009.

[CK76] C. Cheng and G. Knorr. The integration of the vlasov equation in configuration space. *Journal of Computational Physics*, 22(3):330 – 351, 1976.

[CL87] C. Chui and M. Lai. Computation of box-splines and b-splines on triangulations of nonuniform rectangular partitions. *Approx. Theory Appl*, 3:37–62, 1987.

[CL08] P. Chatelain and A. Leonard. Isotropic compact interpolation schemes for particle methods. *Journal of Computational Physics*, 227(6):3244–3259, 2008.

[CLS07] N. Crouseilles, G. Latu, and E. Sonnendrücker. Hermite spline interpolation on patches for parallelly solving the vlasov-poisson equation. *International Journal of Applied Mathematics and Computer Science*, 17(3):335–349, 2007.

[CSSZ03] 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, Technical Report, 2003.

[CVDV06] L. Condat and D. Van De Ville. Three-directional box-splines: characterization and efficient evaluation. *IEEE Signal Process. Lett.*, 13(7):417–420, 2006.

[CVDV07] 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.

[CVDV08] L. Condat and D. Van De Ville. New optimized spline functions for interpolation on the hexagonal lattice. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 1256–1259. IEEE, 2008.

[CVDVU06] 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.

[DB78] C. De Boor. A practical guide to splines. *Mathematics of Computation*, 1978.

[dB90] C. de BOOR. *Quasiinterpolants and approximation power of multivariate splines*. Springer, 1990.

[DBH83] C. De Boor and K. Höllig. Bivariate box splines and smooth pp functions on a three direction mesh. *Journal of Computational and Applied Mathematics*, 9(1):13–28, 1983.

[DBHR93] C. De Boor, K. Höllig, and S. Riemenschneider. *Box Splines*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[DH09] J. Dolbow and I. Harari. An efficient finite element method for embedded interface problems. *International journal for numerical methods in engineering*, 78(2):229–252, 2009.

[DL91] M. Daehlen and T. Lyche. Box splines and applications. In *Geometric Modeling*, pages 35–93. Springer, 1991.

[EDH10] A. Embar, J. Dolbow, and I. Harari. Imposing dirichlet boundary conditions with nitsche's method and spline-based finite elements. *International Journal for Numerical Methods in Engineering*, 83(7):877–898, 2010.

[Far14] G. Farin. *Curves and surfaces for computer-aided geometric design: a practical guide*. Elsevier, 2014.

[FMH04] S. Fernández-Méndez and A. Huerta. Imposing essential boundary conditions in mesh-free methods. *Computer methods in applied mechanics and engineering*, 193(12):1257–1275, 2004.

[FP15] F. Filbet and C. Prouveur. High order time discretization for backward semi-lagrangian methods. https://hal.archives-ouvertes.fr/hal-01133854, 2015.

[FS03] F. Filbet and E. Sonnendrücker. Comparison of eulerian vlasov solvers. *Computer Physics Communications*, 150(3):247–266, 2003.

[FY14] F. Filbet and C. Yang. Mixed semi-lagrangian/finite difference methods for plasma simulations, September 2014. https://hal.inria.fr/hal-01068223.

[GAB+16] V. Grandgirard, J. Abiteboul, J. Bigot, T. Cartier-Michaud, N. Crouseilles, G. Dif-Pradalier, C. Erhlacher, D. Esteve, X. Garbet, P. Ghendrih, et al. A 5d gyrokinetic full-f global semi-lagrangian code for flux-driven ion turbulence simulations. 2016.

[GB05] D. Gurnett and A. Bhattacharjee. *Introduction to Plasma Physics: With Space and Laboratory Applications*. Cambridge University Press, 2005.

[GBB+06] 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.

[GR14] 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.

[GS03] M. Griebel and M. A. Schweitzer. A particle-partition of unity method part v: boundary conditions. In *Geometric Analysis and Nonlinear Partial Differential Equations*, pages 519–542. Springer, 2003.

[GS12] V. Grandgirard and Y. Sarazin. Gyrokinetic simulations of magnetic fusion plasmas. *Panoramas et synthèses (submitted)*, 2012.

[GSG+06] V. Grandgirard, Y. Sarazin, X. Garbet, G. Dif-Pradalier, P. Ghendrih, N. Crouseilles, G. Latu, E. Sonnendrucker, N. Besse, and P. Bertrand. Gysela, a full-f global gyrokinetic semi-lagrangian code for itg turbulence simulations. In *AIP Conference Proceedings*, volume 871, page 100. IOP INSTITUTE OF PHYSICS PUBLISHING LTD, 2006.

[GSR98] 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.

[HCB05] T. J. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering*, 194(39):4135–4195, 2005.

[HMSS15] 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.

[Kal03] E. Kalnay. *Atmospheric Modeling, Data Assimilation and Predictability*. Cambridge university press, 2003.

[Kob97] L. Kobbelt. Stable evaluation of box-splines. *Numerical Algorithms*, 14(4):377–382, 1997.

[KP09] M. Kim and J. Peters. Fast and stable evaluation of box-splines via the bb-form. *Numerical Algorithms*, 50(4):381–399, 2009.

[KYPK15] 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.

[LM97] C. Lacour and Y. Maday. Two different approaches for matching nonconforming grids: The mortar element method and the feti method. *BIT Numerical Mathematics*, 37(3):720–738, 1997.

[LMS08] T. Lyche, C. Manni, and P. Sablonnière. Quasi-interpolation projectors for box splines. *Journal of Computational and Applied Mathematics*, 221(2):416–429, 2008.

[MAGS14] F. Moro, P. Alotto, M. Guarnieri, and A. Stella. Domain decomposition with the mortar cell method. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 27(3):461–471, 2014.

[MCG+98] R. Miller, M. Chu, J. Greene, Y. Lin-Liu, and R. Waltz. Noncircular, finite aspect ratio, local equilibrium model. *Physics of Plasmas (1994-present)*, 5(4):973–978, 1998.

[Mer79] R. M. Mersereau. The processing of hexagonally sampled two-dimensional signals. *Proceedings of the IEEE*, 67(6):930–949, 1979.

[Mit73] 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.

[MMPS16] M. Mehrenberger, L. S. Mendoza, C. Prouveur, and E. Sonnendrücker. Solving the guiding-center model on a regular hexagonal mesh. *ESAIM: Proceedings and Surveys*, 53:149–176, 2016.

[MPS11] C. Manni, F. Pelosi, and M. L. Sampoli. Generalized b-splines as a tool in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 200(5):867–881, 2011.

[MSM+13] 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.

[Nit71] J. Nitsche. Über ein variationsprinzip zur lösung von dirichlet-problemen bei verwendung von teilräumen, die keinen randbedingungen unterworfen sind. In *Abhandlungen aus dem mathematischen Seminar der Universität Hamburg*, volume 36, pages 9–15. Springer, 1971.

[PT97] L. Piegl and W. Tiller. *The NURBS Book (2Nd Ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[Rat11] 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.

[Run01] C. Runge. Über empirische funktionen und die interpolation zwischen äquidistanten ordinaten. *Zeitschrift für Mathematik und Physik*, 46(224-243):20, 1901.

[Sab96] P. Sablonnière. Quasi-interpolants associated with h-splines on a three-direction mesh. *Journal of computational and applied mathematics*, 66(1):433–442, 1996.

[Sab02] P. Sablonnière. *H-splines and quasi-interpolants on a three directional mesh*. Springer, 2002.

[SAM68] 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.

[Sch46] I. J. Schönberg. Contributions to the problem of approximation of equidistant data by analytic functions. *Quart. Appl. Math*, 4(2):45–99, 1946.

[SF05] K. Scherer and A. E. Forschung. *Space Weather: The Physics Behind a Slogan*. Lecture Notes in Physics. Springer, 2005.

[SLP12] J. D. Sanders, T. A. Laursen, and M. A. Puso. A nitsche embedded mesh method. *Computational Mechanics*, 49(2):243–257, 2012.

[SRBG99] 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.

[Ste14] C. Steiner. *Numerical computation of the gyroaverage operator and coupling with the Vlasov gyrokinetic equations*. PhD thesis, IRMA, December 2014.

[TW05] A. Toselli and O. Widlund. *Domain decomposition methods: algorithms and theory*, volume 3. Springer, 2005.

[Uli87] R. Ulichney. *Digital Halftoning*. MIT Press, Cambridge, MA, 1987.

[VDVBU+04] 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.

[ZGB88] 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.

[ZTZ05] O. Zeinkiewicz, R. Taylor, and J. Zhu. The finite element method: its basis and fundamentals, 2005.