

INSTITUT DE RECHERCHE MATHÉMATIQUE AVANCÉE
Université de Strasbourg, CNRS (UMR 7501) et INRIA Grand-Est Nancy
7, rue René Descartes 67084 Strasbourg Cedex

Isogeometric Analysis in Plasma Physics and Electromagnetism

Ahmed RATNANI

Thèse soutenue le 07 octobre 2011 devant le jury composé de

Eric SONNENDRÜCKER	Directeur de thèse
Annalisa BUFFA	Rapporteur
Rémi ABGRALL	Rapporteur
Nicolas CROUSEILLES	Examineur
Philippe HELLUY	Examineur
Boniface NKONGA	Examineur

Isogeometric Analysis in Plasma Physics and Electromagnetism

RATNANI Ahmed

July 13, 2011

Remerciements

A la mémoire de mon père

Résumé

Dans cette thèse, nous nous intéressons à la résolution de divers problèmes numériques issues de la physique des plasmas, en utilisant l'analyse isogéométrique. L'une des principales difficultés que rencontrent les méthodes actuelles est la prise en compte de la complexité de la géométrie. L'analyse isogéométrique permet de répondre à ce type de problèmes, que ce soit par la modélisation exacte d'une grande classe de domaines, ou par la précision des approximations qu'on peut obtenir le cas échéant. Tout d'abord on commence par présenter le contexte physique.

Contexte physique

Les plasmas

Avec les trois états de matières, *i.e* solide, liquide and gaz, le plasma est considéré être le quatrième. Même si on ne l'observe pas dans notre quotidien, il représente plus de 99% de la matière connu à ce jour de l'univers. Le mot grecque Plasma a été introduit pour la première fois par le physiologiste Jan Evangelista Purkinje, au milieu du 19^{ème} siècle, ce qui veut dire en français *moulé*. Une intéressante présentation historique est donnée dans [9].

Jusqu'aux années 1950, la recherche concernant les plasmas confinés était classée secret défense. Mais à partir de 1958, les Etats Unis, la Grande Bretagne et l'Union Soviétique les déclassent, lorsqu'ils se sont rendus compte que ces recherches ne pouvaient en aucun cas avoir d'intérêt militaire. Assez rapidement, la configuration du *Tokamak* développée par les russes, s'impose comme étant la meilleure façon d'atteindre un confinement magnétique. La conception toroidale des *Tokamaks*, ainsi que les lignes de champs magnétiques qui s'enroulent d'une manière hélicoïdale, (voir figure 1), piègent les particules grâce à la force de Lorentz. Ainsi, les particules vont suivre une trajectoire hélicoïdale suivant ces lignes de champs.

La fusion nucléaire et le projet ITER

Le recours à l'énergie nucléaire a été une conséquence du choc pétrolier de 1973. La France, par exemple produit actuellement les trois quarts de son énergie électrique par voie nucléaire. Pour le moment, les réacteurs fonctionnent selon le principe de la fission nucléaire. Celle-ci est très polluante, car consiste à désintégrer le noyau d'un atome lourd (l'Uranium 235) en noyaux plus légers. Le projet ITER (International Thermonuclear Experimental Reactor), est un prototype de réacteur nucléaire à fusion, actuellement en construction à Cadarache, et qui aura une puissance de 500MW. Le but de ce projet étant de démontrer la faisabilité du point scientifique et technique de la fusion nucléaire.

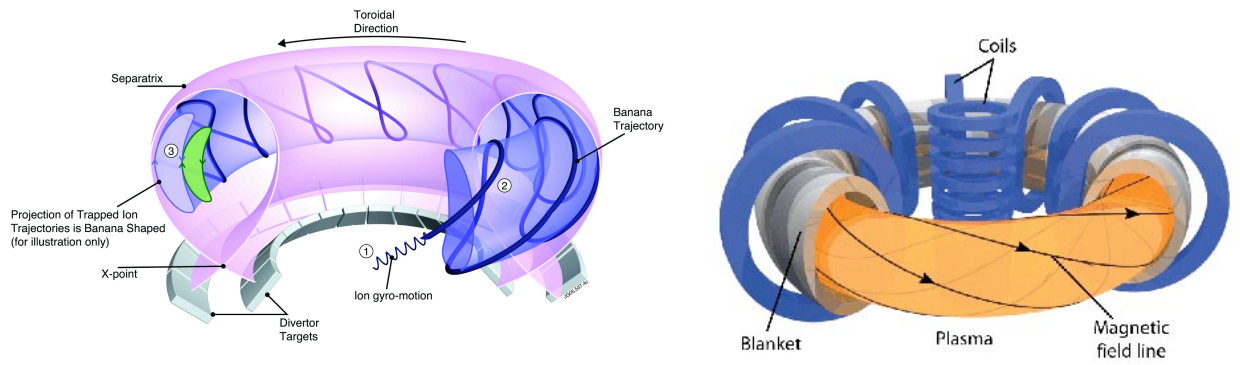


Figure 1: Helical magnetic field lines

Afin d'aboutir aux technologies nécessaires et suffisantes au développement du réacteur expérimental *DEMO*, dont la puissance sera de 1500MW , à une fin industrielle.

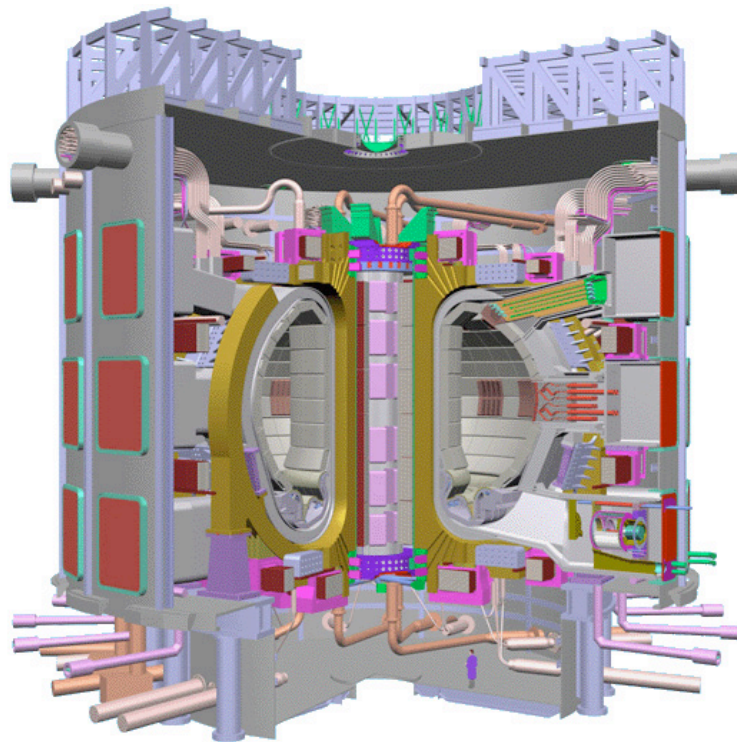


Figure 2: ITER tokamak

Le confinement magnétique

Comment peut-on s'assurer que les particules restent rassemblées dans le tokamak? Pour le soleil, tout est *simple*, la force gravitationnel se charge de cette tâche. Sur terre, la structure des tokamaks, couplée à de fort champs magnétiques, permettent de confiner les particules, qui se verront obliger de bouger autour de ces lignes de champs tout en

suivant un mouvement hélicoïdal.

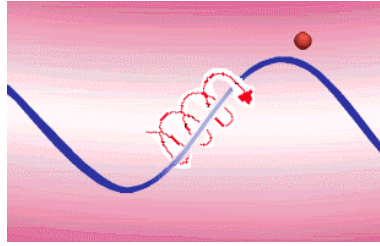


Figure 3: A confined particle follows the magnetic field line, while gyrating

La figure 3 montre la trajectoire d'une particule confinée. Dans ce cas, la direction de la trajectoire va dépendre du signe de la charge. Se pose alors la question de la nature des lignes de champs magnétiques. Pour assurer le confinement dans la 3^{eme} direction, les lignes de champs ne peuvent être purement toroidales; sinon, elles vont dériver (voir figure 4).

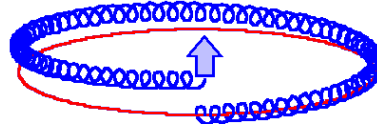


Figure 4: In purely toroidal configuration, magnetic field lines drift in the third dimension

Une autre conséquence d'un champ magnétique purement toroïdal est son caractère non-uniforme. En effet, un champ magnétique axisymétrique généré par un solénoïde coudé, varie inversement avec le rayon majeur.

Pour éviter ce genre de problèmes, on impose un fort champ magnétique poloidal. Ce qui nous emmène à la forme suivante du champ magnétique:

$$\mathbf{B} = \mathbf{B}_P + \mathbf{B}_T \quad (0.0.1)$$

où \mathbf{B}_P est la composante poloidale du champ, et \mathbf{B}_T la composante toroidale.

Conséquence, les lignes de champ magnétiques vont se tordre à cause de la composante poloidale (voir figure 5). Pour plus de détails, voir [96].

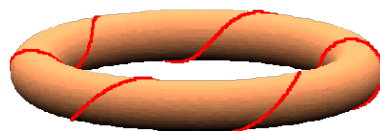


Figure 5: Twisted magnetic field line, under the poloidal component

Le système de coordonnées

Il existe différents systèmes de coordonnées utilisés en physique des plasmas. Dans la figure 6, on montre le lien entre les coordonnées toroïdales (r, θ, ϕ) et cylindriques (R, z, ϕ) .

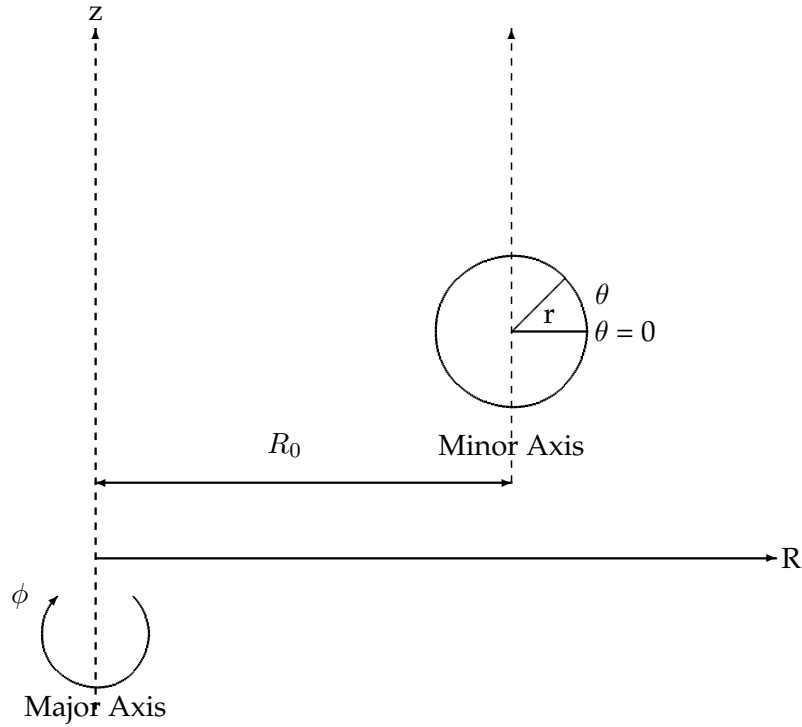


Figure 6: Toroidal (r, θ, ϕ) and cylindrical (R, z, ϕ) coordinate systems.

Les principaux modèles physiques

Trois paramètres caractérisent le plasma:

1. la densité des particules n
2. la température pour chaque espèce, T_i pour les ions, et T_e pour les électrons,
3. le champ magnétique stationnaire B .

L'interaction entre les particules et le champ électromagnétique, détermine la dynamique du plasma. Soit $\mathbf{x}_p^i(t)$ la position d'un ion à l'instant t , et $\mathbf{x}_p^e(t)$ celle de l'électron. Ces particules vont générer un courant, qui une fois injectées dans les équations de Maxwell, mettent à jour le champ électromagnétique. Ces nouvelles valeurs des champs vont mettre à jour à leur tour, la position et la vitesse de chaque particule grâce à la force de Lorentz.

A cause de la géométrie, un système de coordonnées typique serait,

$$\mathbf{x} = (r, \theta, \varphi) \quad (0.0.2)$$

pour les positions des particules. Et pour les vitesses:

$$\mathbf{v} = (v_{\parallel}, v_{\perp}, \alpha) \quad (0.0.3)$$

Un modèle naïf prendrait en compte des inconnues $6D$. Cependant, à cause du nombre très élevé de particules à l'intérieur du *tokamak*, ce serait complètement illusoire de vouloir utiliser un modèle à *n-corps*, prenant en compte les particules et leurs contributions pour le champ électromagnétique. Par contre, à partir d'une description *n-corps*,

nous pouvons dériver des modèles de plus en plus simples, le prix à payer évidemment est la perte d'informations. Le nouveau modèle sera restreint alors à certaine configuration du plasma. Dans la figure 7, nous montrons une hiérarchie de ces modèles (la liste est non exhaustive), selon la physique, et en fonction de la complexité du modèle.

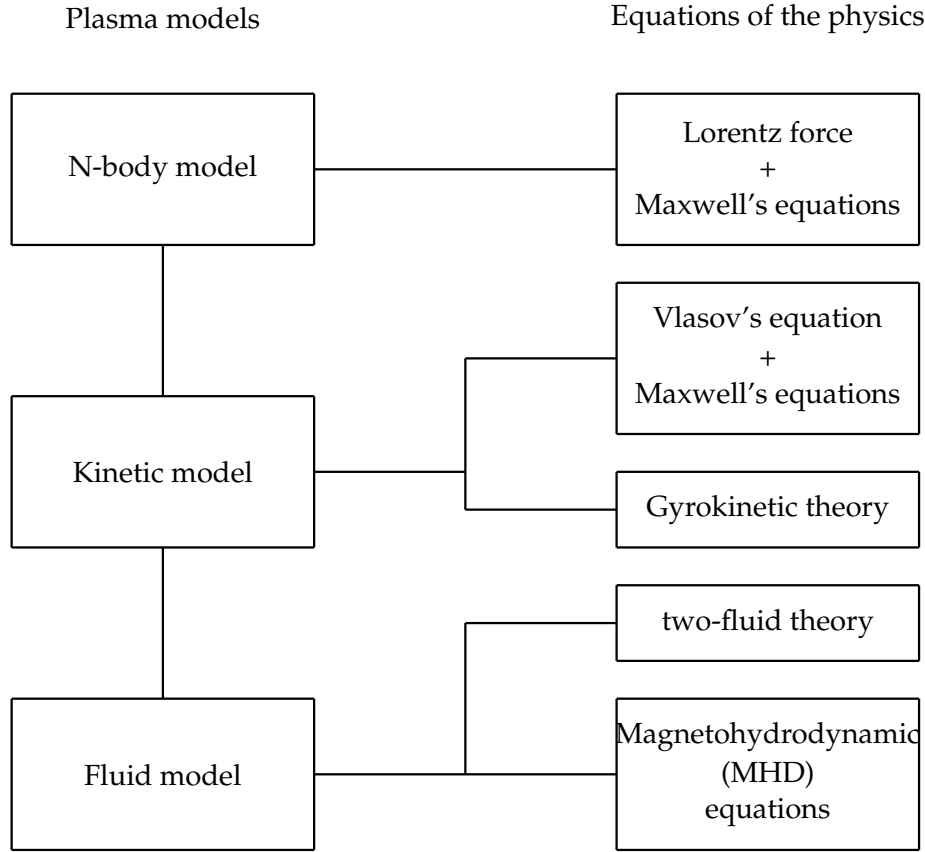


Figure 7: Plasma models

Modèle de Vlasov-Maxwell

Dans ce type de modèle, on étudie l'évolution des distributions des positions et vitesses des particules. L'équation de Vlasov est une équation de transport. Dans le cas où on néglige les collisions, et l'agitation thermique,

$$(\partial_t + \mathbf{v} \cdot \nabla_x) f + \frac{q}{m} (\mathbf{E} + \mathbf{v} \times \mathbf{B}) \nabla_v f = 0 \quad (0.0.4)$$

où f est la fonction de distribution des ions ou électrons. A ces particules nous associons la densité de charge électrique ρ , et la densité de courant électrique \mathbf{J} :

$$\rho = \sum_s q_s \int f_s(\mathbf{x}, \mathbf{v}, t) d\mathbf{v} \quad (0.0.5)$$

$$\mathbf{J} = \sum_s q_s \int f_s(\mathbf{x}, \mathbf{v}, t) \mathbf{v} d\mathbf{v} \quad (0.0.6)$$

Ces densités sont couplées aux équations de Maxwell:

$$\partial_t \mathbf{E} - \text{rot } \mathbf{H} = -\mathbf{J}, \quad (0.0.7)$$

$$\partial_t \mathbf{H} + \text{rot } \mathbf{E} = 0, \quad (0.0.8)$$

$$\text{div } \mathbf{E} = \rho, \quad (0.0.9)$$

$$\text{div } \mathbf{H} = 0 \quad (0.0.10)$$

Il existe deux principales méthodes numériques pour la simulation de tels modèles. La première est dite particulière, Particle In Cell (PIC) . La seconde est une méthode Eulerienne.

Néanmoins, notre modèle comporte toujours des inconnues $6D$. Pour réduire ce modèle, tout en gardant le caractère statistique, on utilise la gyromoyenne. La théorie gyrocinétique développée par Brizard [3], permet d'étudier des inconnues $5D$.

Modèle deux fluides

Le modèle deux fluides est intermédiaire à la théorie de Vlasov et la magnétohydrodynamique (MHD). Le plasma y est décrit comme un système de deux fluides (ions d'un côté et électrons de l'autre) en interaction. Pour plus de détails (*c.f* [71, 9]).

Magnétohydrodynamique

Il s'agit du modèle le moins détaillé. Le plasma y est décrit comme un seul fluide électrique conducteur. Pour plus de détails, *c.f* [71, 9, 29].

Présentation de la thèse

Dans cette thèse, nous avons essayé de considérer plusieurs approches communément utilisées dans la physique des plasmas. Notre but n'était pas d'étudier tous les modèles, mais plutôt de révéler l'intérêt de l'analyse isogéométrique. Ce manuscrit ne respecte absolument pas l'ordre chronologique du travail effectué durant cette thèse. Nous avons essayé d'écrire un document auto-suffisant; faute de temps la physique sera moins détaillée, et on notera les rappels des principales bases dans chaque chapitre d'application. La difficulté de rédaction, mais aussi de l'approche, est qu'elle s'inscrit à l'intersection de plusieurs domaines, tout aussi riches et passionnants: la physique des plasmas, l'analyse numérique, la CAO, et l'implémentation informatique. Vouloir couvrir la totalité de cette interaction serait complément illusoire. Néanmoins, notre but était d'essayer de rendre un document qui permet à une personne qui vient de l'un de ces domaines d'avoir les bases nécessaires pour aborder l'ensemble des problèmes traités. Nous rendant compte de l'importance du code informatique à développer, nous avons pu achever cette thèse avec une librairie *PyIGA* qui permet la résolution des équations aux dérivées partielles en utilisant l'approche IGA (*c.f* annexe D pour plus de

détails).

Remark 0.0.1 *Il est extrêmement important de noter que l'une de principales propriétés de l'analyse isogéométrique est que la description du domaine est donnée par les mêmes fonctions de bases qu'on utilisera pour approcher les solutions des équations aux dérivées partielles qu'on traite. Cette description reste exacte après raffinement; ce qui explique le terme **iso**-géométrique. Il se trouve que dans la majorité des applications (en physique des plasmas) qu'on va considérer, on n'aura pas de description exacte du domaine. Cette description sera, en général, une approximation. Dans ce cas là, il s'agira plutôt d'une méthode isoparamétrique. Le lecteur devra faire attention à ce point particulier. On utilisera la version **isoparamétrique** de l'IGA lorsque la physique l'impose. Sinon, lorsque la description du domaine est exacte, on utilisera la version classique de l'IGA.*

Nous présentons brièvement le contenu de chaque chapitre.

Chapitre 1 : Introduction aux splines

Ce chapitre est constitué de rappels que nous considérons assez importants pour pouvoir entamer les études que nous voudrions faire. Nous présentons la définition des splines dans le cadre théorique, ainsi que les principales propriétés d'approximations dont on aura besoin par la suite. On présentera aussi les différents algorithmes pour les évaluer, mais aussi pour manipuler les courbes *B-splines*, en tout cas pour ce dont on a besoin pour les problèmes que nous traitons. Nous présentons aussi la méthode des web-splines développée par Höllig [58], que nous avons étudiée tout au début de cette thèse. Nous montrons les limites de cette méthode, en tout cas pour les problèmes qu'on voudra traiter. Ensuite, nous introduisons l'analyse isogéométrique en rappelant les principales propriétés d'approximation. L'idée de base de cette approche est d'utiliser les fonctions de base

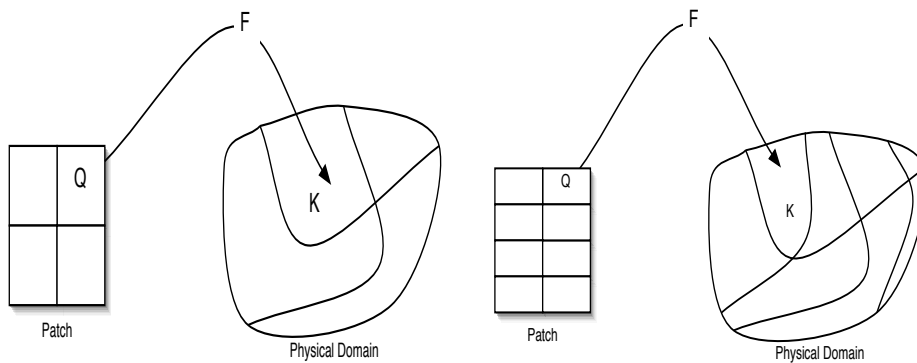


Figure 8: Mapping from the patch to the physical domain: (left) initial patch, (right) patch after h-refinement in the η direction. Here, we have $K = \mathbf{F}(Q)$

Chapitre 2 : Etude des équations aux dérivées partielles elliptiques

Dans ce chapitre on montre pas à pas la résolution des équations différentielles elliptiques. Pour la validation numérique, nous avons étudié divers domaines, construits à l'aide des *B-splines* ou *NURBS*. Nous donnons aussi une solution analytique pour

l'équation de Poisson dans un domaine général. Nous terminons ce chapitre par traiter des équations non-linéaires.

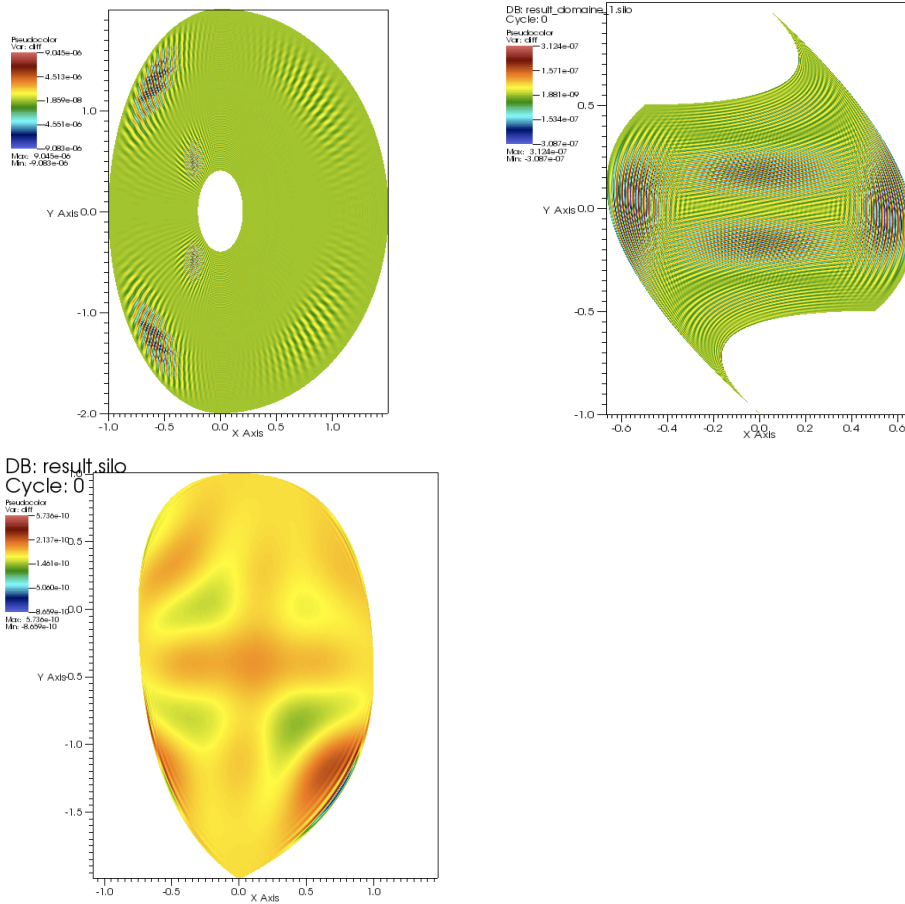


Figure 9: The difference between the numerical and analytical solution

Chapitre 3 : Application à l'équation de Quasi-Neutralité

Dans ce chapitre nous traitons l'équation de Quasi-Neutralité, l'équivalent de l'équation de Poisson dans le cadre de la théorie gyrocinétique. Nous commençons par traiter le problème sur une couronne. Nous montrons que dans ce cadre on peut utiliser une approche basée sur des FFT pour la résolution du système linéaire. On s'inspirera de cette méthode pour développer plus tard l'approche Analyse Isogéométrique rapide, Fast-IGA. Ensuite, on utilise l'analyse isogéométrique pour résoudre le problème dans le cadre générale, pour des solutions dites de turbulence. Les résultats seront comparés avec une méthode d'éléments finis basée sur des triangles. Le code développé dans cette partie est en cours de couplage avec *Gysela*, en vue d'une future probable intégration.

Chapitre 4 : Applications aux équations de Maxwell en 2D

Dans ce chapitre nous résolvons les équations de Maxwell en 2D en se basant sur un diagramme de DeRham [16, 18]. Dans une formulation **H-div**, un choix judicieux de la

Spline degree	FIGA	SPLU	Spline degree	FIGA	SPLU
1	0.012	0.013	1	0.008	0.38
2	0.014	0.046	2	0.013	3.81
3	0.014	0.073	3	0.012	10.69
4	0.013	0.098	4	0.016	19.17
5	0.015	0.124	5	0.017	31.95
6	0.015	0.152	6	0.020	47.01
7	0.015	0.179	7	0.023	65.00

Figure 10: CPU-time, in seconds, spent in solving (left) and initializing (right) the linear system, using the new approach, namely Fast IGA, compared to SuperLU. Test done on a grid 128×128

Spline degree	FIGA	SPLU	Spline degree	FIGA	SPLU
1	0.074	0.067	1	0.021	3.38
2	0.076	0.967	2	0.043	31.40
3	0.075	3.505	3	0.052	197.31
4	0.075	16.070	4	0.060	330.28
5	0.077	32.852	5	0.069	415.63

Figure 11: CPU-time, in seconds, spent in solving (left) and initializing (right) the linear system, using the new approach, namely Fast IGA, compared to SuperLU. Test done on a grid 256×256

base nous permet de transformer les dérivationes (l'opérateur rotationnel par exemple) en opérations algébriques, se rapprochant à des différences directionnelles dans le cadre des méthodes de Différences Finies. Ainsi, à chaque itération, on aura besoin d'inverser une seule matrice. Nous avons aussi implémenté la formulation **H-rot**. Nous avons aussi considéré le cadre axisymétrique, et nous avons montré que l'on n'obtient pas l'ordre de convergence classique pour la formulation **H-div**, alors qu'on le retrouve dans le cadre d'une formulation **H-rot**.

Chapitre 5 : Un solveur PIC axisymétrique basé sur l'analyse isogéométrique

Dans ce chapitre nous résolvons le système Vlasov-Maxwell en coordonnées axisymétriques. Les résultats sont en cohérence avec l'expérience. Le code développé devra être parallélisé pour de meilleures performances.

La principale nouveauté dans ce travail est que les particules vivent dans le patch. On utilise alors le mapping **F** pour transformer les positions des particules ainsi que leurs vitesses. On a calculé les équations de mouvements des particules sur le patch (voir le chapitre C), dont nous donnons ici les grandes lignes.

Les équations du mouvement des électrons, qui ont une expression simple en géométrie cartésienne :

$$\begin{cases} \frac{d\mathbf{X}}{dt} = \mathbf{V}, \\ \frac{d\mathbf{V}}{dt} = -(\mathbf{E} + \mathbf{V} \wedge \mathbf{B}), \end{cases}$$

doivent être écrites en géométrie $2D$ axisymétrique dans un système de coordonnées quelconque.

Pour cela, on passe des coordonnées cartésiennes (x, y, z) aux coordonnées cylindriques (z, r, θ) , puis axisymétriques (z, r) puis à un système de coordonnées quelconque (ξ, η) . On écrit le Lagrangien (voir [110] et [49]) dans ces coordonnées :

$$L = \frac{1}{2}m(M_\xi \dot{\xi}^2 + M_\eta \dot{\eta}^2 + 2M_{\xi\eta} \dot{\xi}\dot{\eta}) + e(A_\xi \dot{\xi} + A_\eta \dot{\eta} - \phi),$$

où

$$\begin{aligned} M_\xi &= \left(\frac{\partial r}{\partial \xi}\right)^2 + \left(\frac{\partial z}{\partial \xi}\right)^2, \\ M_\eta &= \left(\frac{\partial r}{\partial \eta}\right)^2 + \left(\frac{\partial z}{\partial \eta}\right)^2, \\ M_{\xi\eta} &= \frac{\partial r}{\partial \xi} \frac{\partial r}{\partial \eta} + \frac{\partial z}{\partial \xi} \frac{\partial z}{\partial \eta}, \end{aligned}$$

avec $\mathbf{A} = (A_\xi, A_\eta)$ le potentiel vecteur et ϕ le potentiel scalaire : $\mathbf{E} = -\frac{\partial \mathbf{A}}{\partial t} - \nabla \phi$.

Les équations d'Euler-Lagrange

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\mathbf{q}}} = \frac{\partial L}{\partial \mathbf{q}},$$

où

$$\mathbf{q} = \begin{pmatrix} \xi \\ \eta \end{pmatrix},$$

conduisent aux équations du mouvement suivantes :

$$\left\{ \begin{aligned} & \det(J) \frac{d\dot{\xi}}{dt} + \dot{\xi}^2 K_{\xi,\eta} + \dot{\eta}^2 K_{\eta,\eta} + 2\dot{\eta}\dot{\xi} K_{\eta\xi,\eta} \\ &= -\frac{1}{\det(J)} \left(((\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B})|_\xi) M_\eta - (\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B})|_\eta M_{\xi\eta} \right), \\ & \det(J) \frac{d\dot{\eta}}{dt} - \dot{\xi}^2 K_{\xi,\xi} - \dot{\eta}^2 K_{\eta,\xi} - 2\dot{\eta}\dot{\xi} K_{\xi\eta,\xi} \\ &= -\frac{1}{\det(J)} \left(((\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B})|_\eta) M_\xi - (\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B})|_\xi M_{\xi\eta} \right) \end{aligned} \right.,$$

où tous les coefficients sont explicités en annexe, tout comme les détails des calculs. Ce travail a été effectué dans le cadre du Cemracs 2010 [8], avec A. Back, A. Crestetto, et E. Sonnendrücker.

Chapitre 6 : Applications aux méthodes semi-lagrangienne

Dans ce chapitre on étudie une méthode semi-lagrangienne couplée avec des mappings *B-splines* pour transformer une grille cartésienne (sur le patch) en une grille sur le domaine réel. On traitera alors le système Vlasov-Poisson en 2D, sur des domaines

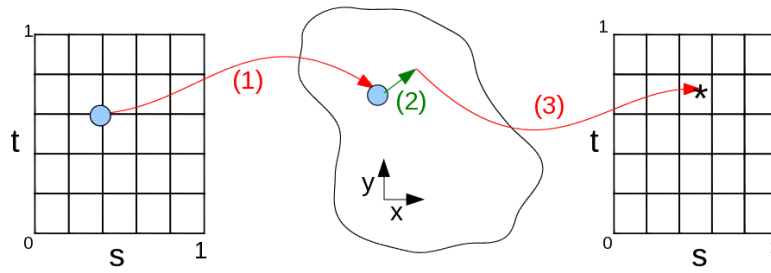


Figure 12: The Semi-Lagrangian method in complex geometry: (1) map the position in the reference space into physical space, (2) follow the characteristic backwards in physical space, (3) map the obtained position back in the reference space to perform the interpolation

complexes (voir figure 12). Même si le coût numérique est élevé, il peut être considérablement réduit par une architecture du code plus adaptée et parallèle. Les résultats concluants de ce travail, pourront faire l’objet d’une extension du code *Gysel*.

Ce travail a été effectué dans le cadre du Cemracs 2010 [1], avec J. Abiteboul, G. Latu, V. Grandgirard, E. Sonnendrücker et A. Strugarek.

Chapitre 7 : Applications à la MHD

Dans ce chapitre, nous étudions quelques problèmes issus de la MHD. Après l’étude de diffusion fortement anisotropique (voir figure 13), nous traitons une question assez cruciale en physique des plasma; il est communément admis dans la communauté des physiciens d’utiliser les coordonnées curvilignes pour traiter le cas d’un confinement avec un fort champ magnétique. Le but est alors de pouvoir créer des maillages qui suivent les lignes de champs. Evidemment, dans le cadre de l’analyse isogéométrique il n’y a pas de “maillage réels”, l’idée est alors de prendre des surfaces de niveaux du jacobien de la transformation F , qui correspondront à des surfaces fermées reflétant au mieux les surfaces magnétiques (voir figure 14).

Nous traitons aussi le cas de la MHD résistive et incompressible, dans le cadre du test *Current Hole*. Dans ce cas, il faut utiliser des grilles très raffinées pour pouvoir capturer l’évolution du champ magnétique. Le raffinement local se révèle d’une grande nécessité dans ce cadre. Pour le moment, *PyIGA* ne prend pas en compte ce type de raffinement.

Chapitre 8 : Un schéma de DeRham basé sur les Box-splines

Dans ce chapitre, nous explorons de nouvelles pistes pour utiliser les splines sur triangles. En effet, on suivra l’idée développée par Buffa et al [16, 18], pour générer un diagramme de DeRham basés sur ce type d’éléments. Pour le moment, on ne prend pas les conditions aux bord, et donc l’étude est faite plutôt localement. Il reste aussi à implémenter ce type d’éléments (voir la figure 15 pour un exemple de support).

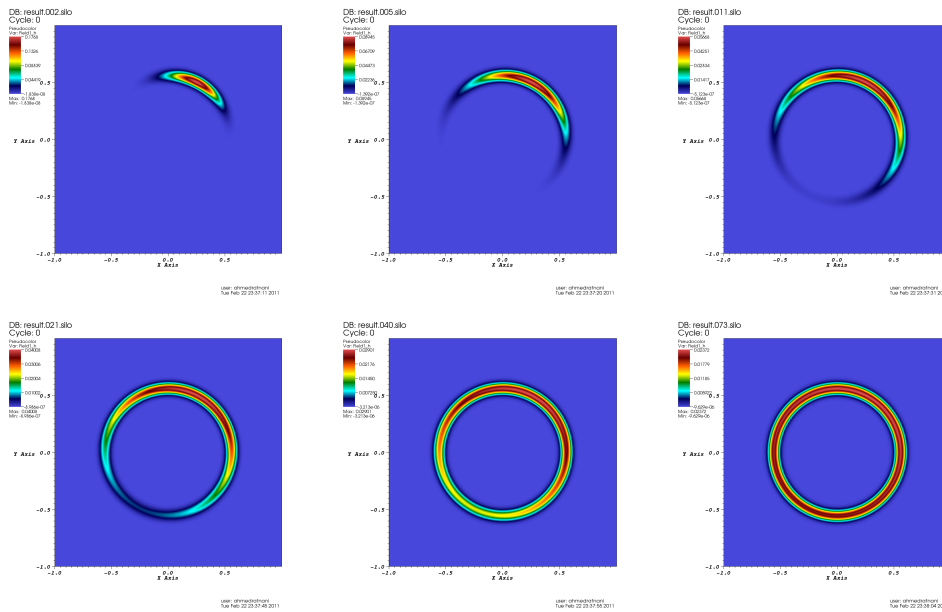


Figure 13: Evolution of the pulse, for a radial section, on a square domain

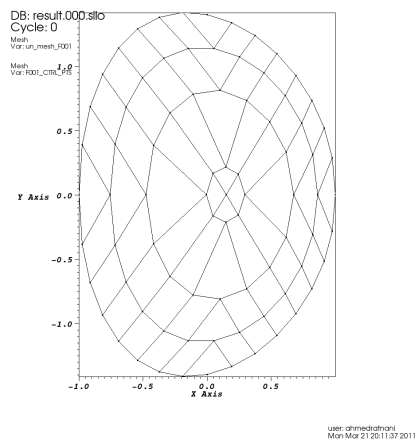


Figure 14: Soloviev solution, example of aligned meshes

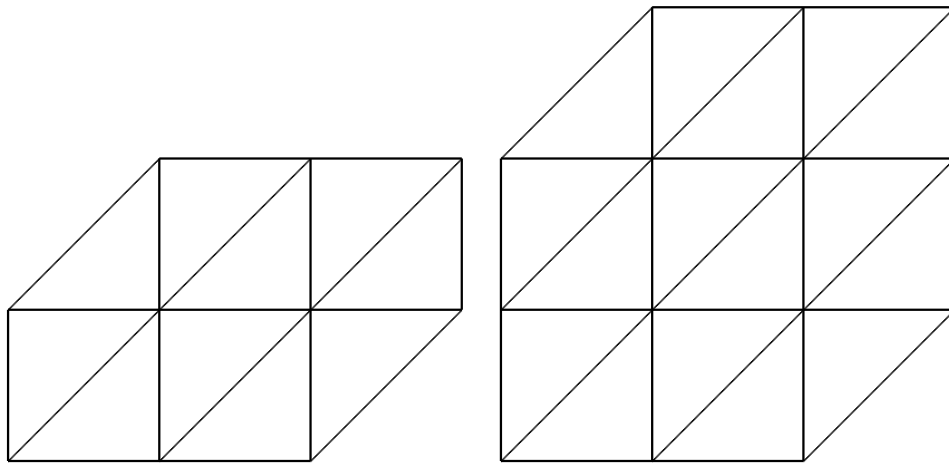


Figure 15: Les supports des box-splines B_{211} et B_{221} .

Publications

Les travaux présentés dans cette thèse ont fait l'objet des publications suivantes:

1. Arbitrary High-Order Spline Finite Element Solver for the Time Domain Maxwell equations (with E. Sonnendrücker). *Journal of Scientific Computing*, 2011, pages 1-20.
2. An Isogeometric Analysis Approach for the study of the gyrokinetic quasi-neutrality equation (with E. Sonnendrücker and N. Crouseilles). *Submitted*.
URL : <http://hal.inria.fr/inria-00587009>
3. Solving the Vlasov equations in complex geometries. (with J. Abiteboul, G. Latu, V. Grandgirard, E. Sonnendrücker and A. Strugarek.). Proceedings of CEMRACS 2010. *Submitted*.
4. An Axisymmetric PIC code based on Isogeometric Analysis (with A. Back, A. Crestetto, and E. Sonnendrücker). Proceedings of CEMRACS 2010. *Submitted*.

Les articles suivants sont cours de préparation,

1. Simulation of 2D reduced MHD using Isogeometric Analysis (with E. Sonnendrücker). *In preparation*.
2. The Fast IGA Approach. *In preparation*.
3. *PyIGA*, Isogeometric Analysis simulations in *Python*. *In preparation*.

Logiciels

Nous terminons la présentation de ce manuscrit, en introduisant les codes développés:

- **WEBSPLINE2D** est un code Fortran pour la résolution des équations aux dérivées partielles en 2D en utilisant la méthode Web-splines. (9'700 lignes)
- **ISOBOX** est un code Fortran. C'est le noyau de *PyIGA*. Il contient un module pour la *C.A.O.*, qui permet d'effectuer les opérations de bases pour manipuler des courbes *NURBS*. Il contient aussi un module pour manipuler les matrices *sparse*. Il contient aussi un module pour la visualisation à l'aide de *VisIt* sous format *sil*. (15'000 lignes)
- **ISOPIC** est un code Fortran, basé sur *ISOBOX*, qui permet la résolution du système Vlasov-Maxwell, en coordonnées axisymétriques, en utilisant l'approche IGA. Les particules vivent dans le patch. (8'900 lignes)
- **PyIGA** est un code Python-Fortran, basé sur *ISOBOX*, qui permet la résolution d'une large catégorie d'edps (système d'edps), en utilisant l'approche IGA. Elle contient aussi quelques familles de splines, généralisations des *B-splines* qui sont les *GB-splines*. (7'200 lignes)
- **PyIGA - GUI** est une interface graphique permettant la création de domaines 2D, ainsi que leur maillages, puis l'exportation en format *XML*. (2'900 lignes)

La librairie *PyIGA* sera présentée plus en détail dans le chapitre D. Il s'agit d'un code écrit en **Fortran** et **Python**. Il a été développé pour faciliter la résolution des équations aux dérivées partielles, où l'utilisateur peut créer les opérateurs dont il a besoin. Ensuite, il fait appel à *scipy* pour le traitement des matrices ainsi obtenues. L'utilisateur peut continuer à communiquer avec la librairie pour les différents diagnostics ou d'éventuels nouveaux assemblages de matrices.

Introduction

Plasmas

Introduction

With the three state of matter, *i.e* solid, liquid and gas, plasma is known to be the fourth one. Even if we don't see such state in our daily lives, it represents much more than 99% of the known matter. The Greek word plasma was introduced by the physiologist Jan Evangelista Purkinje, in the mid nineteenth century, which means *molded*. An interesting history is given in [9].

Until 1950's, researches on plasma confinement were classified as secret defense. However, in 1958, United States of America, Britain, and the then Soviet Union, declassified their researches when they found that controlled fusion research was not of interest in the military domain. Quickly, the *Tokamak* configuration developed by russians, began to take place as the best way to perform a magnetic confinement. The toroidal design of *tokamaks*, and the magnetic field lines that move around it in helical shape (see figure 16), trap particles thanks to the Lorentz force. Particles will follow helical paths around the field lines.

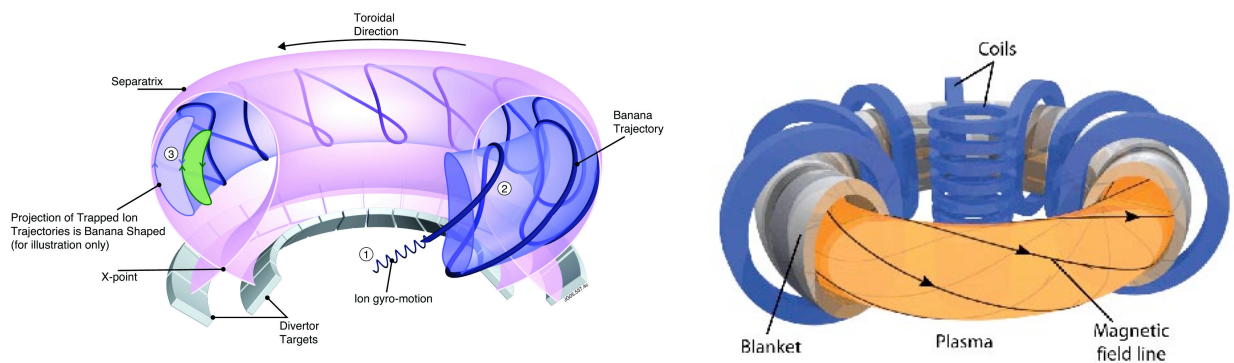


Figure 16: Helical magnetic field lines

ITER Project

The use of Nuclear energy, was a consequence of the 1973 oil crisis. Nowadays, France produces 75% using this energy source. The major disadvantage of such source, is the use of nuclear fission, which leads to important radioactive wastes. In this context, the ITER (International Thermonuclear Experimental Reactor) project, is an interesting way

of generating energy using a nuclear fusion rather than fission one. The main goal of this project, is to prove the feasibility, of nuclear fusion providing a power of $500MW$. This will lead to the construction of an experimental reactor, namely *DEMO*, which will provide a power of $1500MW$, for an industrial use.

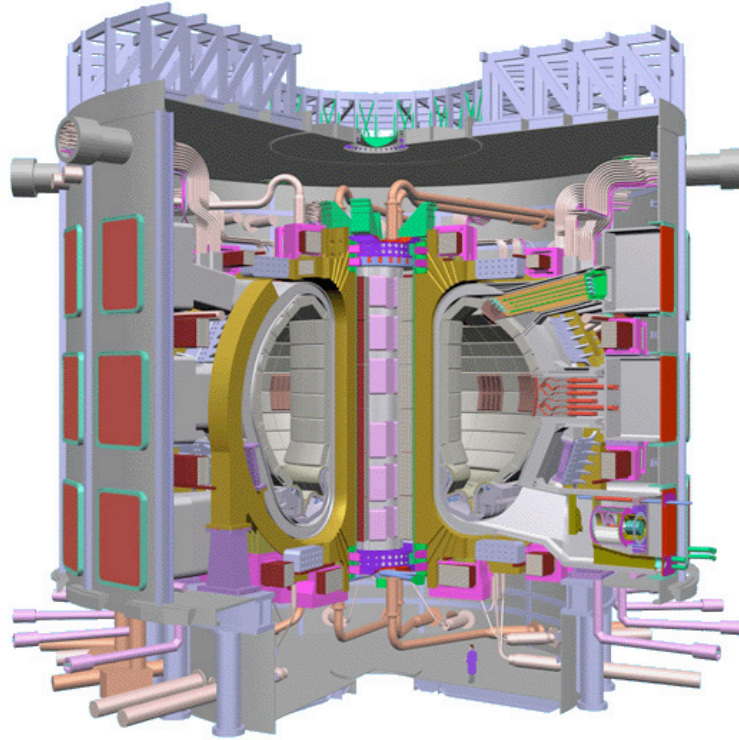


Figure 17: ITER tokamak

Plasma Confinement

How to gather particles? In the sun, it is done thanks to the gravitational force, but this would not be the case on earth! The solution is to use a strong magnetic field, to confine particles. Figure 18 shows the trajectory of a confined particle. In this case, the

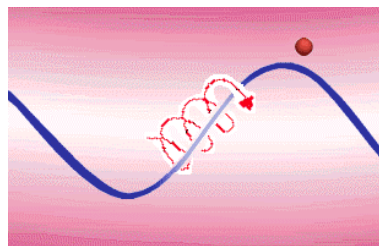


Figure 18: A confined particle follows the magnetic field line, while gyrating

particle will follow the magnetic field line; the direction of its trajectory will depend on the sign of the charge. This poses the question of the shape of the magnetic field lines. To

ensure the confinement in the third dimension, magnetic fields can not be purely toroidal; otherwise magnetic fields lines will drift (see figure 19). Another consequence of purely

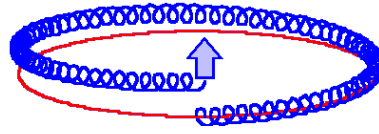


Figure 19: In purely toroidal configuration, magnetic field lines drift in the third dimension

toroidal magnetic field is the non-uniform character of the magnetic field. In fact, an axisymmetric toroidal magnetic field, formed by a bent solenoid, varies inversely with the major radius. To avoid such problem, we impose a strong poloidal magnetic field. Therefore, the magnetic field must be of the form :

$$\mathbf{B} = \mathbf{B}_P + \mathbf{B}_T \quad (0.0.11)$$

where \mathbf{B}_P is the poloidal component of the magnetic field, and \mathbf{B}_T the toroidal one. As a consequence, the magnetic field lines will twist because of the poloidal component (see figure 20). More details can be found in [96].

Another interpretation of this consequence is that the poloidal component \mathbf{B}_P will impose a poloidal motion on the guiding center. In such configuration, the magnetic field

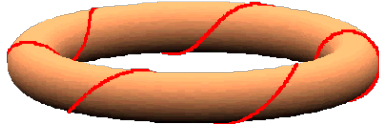


Figure 20: Twisted magnetic field line, under the poloidal component

line will perform a certain number of circuits along the toroidal axis and then loops again. This number is called the safety factor "q". A simple approximation for q, is the safety factor for the cylindrical torus "q_c" :

$$q \simeq q_c = \frac{r}{R_0} \frac{B_T}{B_P}$$

System of coordinates

There exist many system of coordinates that we use in Plasma Physics. In figure 21, we show the link between the toroidal (r, θ, ϕ) and cylindrical (R, z, ϕ) coordinates.

Plasma Models

There are three parameters that characterize a plasma:

1. the particle density n
2. temperature for each species T_i for ions, and T_e for electrons,
3. the steady state magnetic field B .

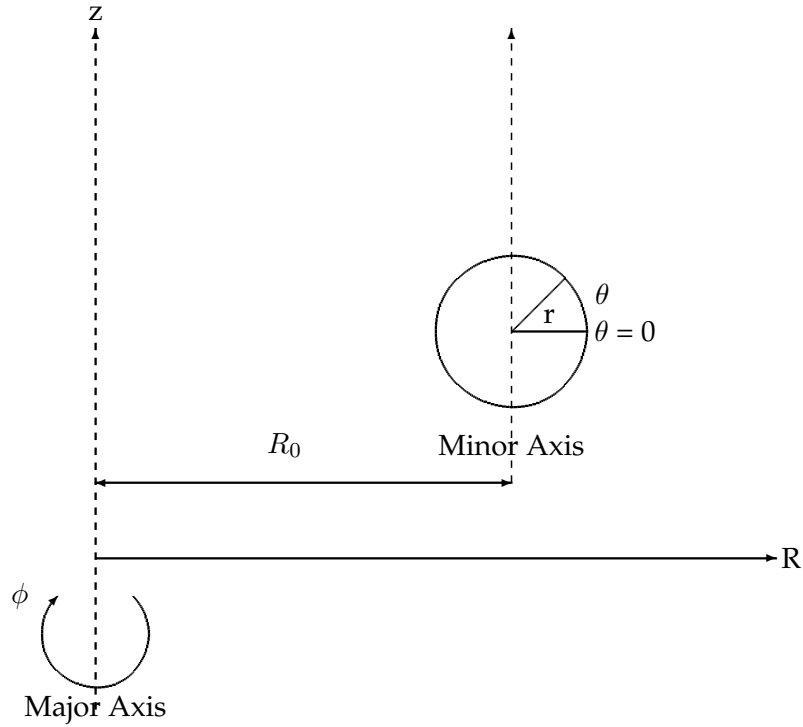


Figure 21: Toroidal (r, θ, ϕ) and cylindrical (R, z, ϕ) coordinate systems.

The interaction between particles and electromagnetic fields, determines the plasma dynamics. Let $\mathbf{x}_p^i(t)$ be the position of an ion particle at the time t , and $\mathbf{x}_p^e(t)$ for electron. Let us also define $\mathbf{v}_p^i(t)$ the velocity of an ion particle at the time t , and $\mathbf{v}_p^e(t)$ for electron. These particles generate a current that injected in Maxwell's equations updates the electromagnetic fields. On the other hand, those new values of the electromagnetic fields are used to update the position and the velocity of particles thanks to the Lorentz force. Because of the geometry, a typical coordinates system would be :

$$\mathbf{x} = (r, \theta, \phi) \tag{0.0.12}$$

for particles position. For the velocity:

$$\mathbf{v} = (v_{\parallel}, v_{\perp}, \alpha) \tag{0.0.13}$$

A naive model, would involve $6D$ unknowns. Because of the big number of particles living in a *tokamak*, it would be obvious to use the *N-body* model, involving all particles and their contribution to the electromagnetic fields. This model is not used in practice. However, we can derive some other models, to reduce parameters, while paying the price of losing some information. The new model will be restricted to some plasma configurations. In figure 22, we show a hierarchical list of models, with the underlying physics, depending on the complexity of the model.

Vlasov Model

In the Vlasov model, rather than studying all particles one by one, we study the evolution of ions/electrons and their velocities distribution functions. The Vlasov equation is

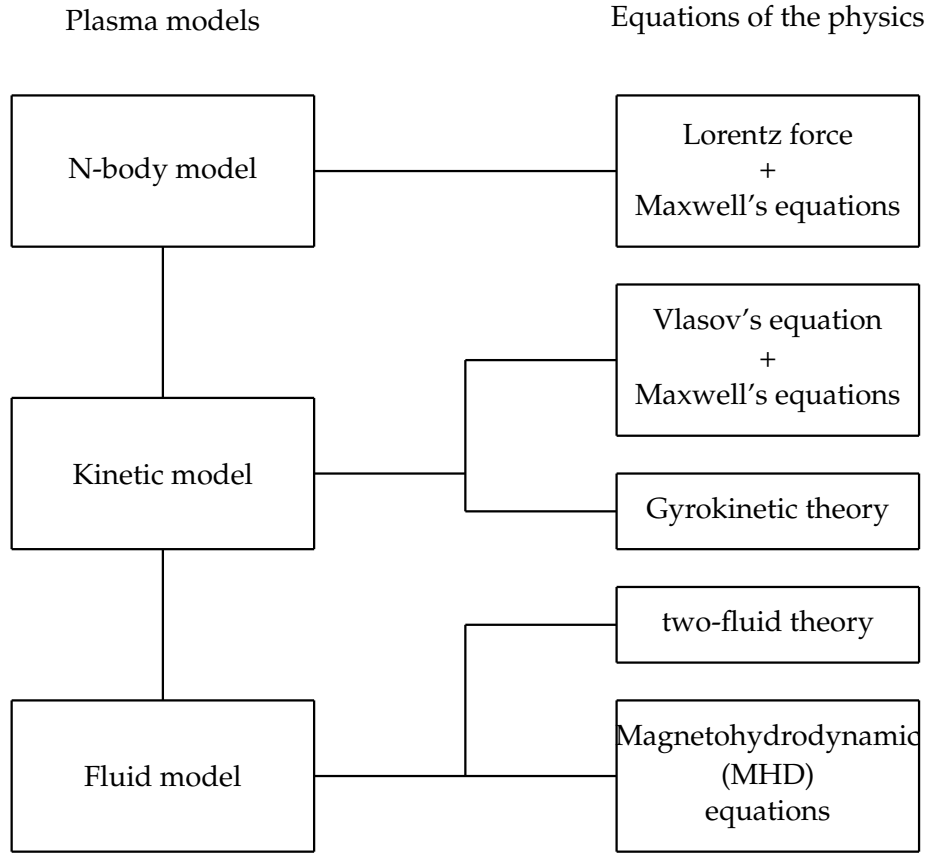


Figure 22: Plasma models

nothing else than a transport equation :

$$(\partial_t + \mathbf{v} \cdot \nabla_x) f + \frac{q}{m} (\mathbf{E} + \mathbf{v} \times \mathbf{B}) \nabla_v f = 0 \quad (0.0.14)$$

where f denotes the distribution function of either ions or electrons.

To these particles, will be associated an electric charge density ρ , and an electric current density \mathbf{J} :

$$\rho = \sum_s q_s \int f_s(\mathbf{x}, \mathbf{v}, t) d\mathbf{v} \quad (0.0.15)$$

$$\mathbf{J} = \sum_s q_s \int f_s(\mathbf{x}, \mathbf{v}, t) \mathbf{v} d\mathbf{v} \quad (0.0.16)$$

These densities are coupled to Maxwell's equations:

$$\partial_t \mathbf{E} - \text{rot } \mathbf{H} = -\mathbf{J}, \quad (0.0.17)$$

$$\partial_t \mathbf{H} + \text{rot } \mathbf{E} = 0, \quad (0.0.18)$$

$$\text{div } \mathbf{E} = \rho, \quad (0.0.19)$$

$$\text{div } \mathbf{H} = 0 \quad (0.0.20)$$

There are two major ways to solve numerically this model. The first one, uses the well known Particle In Cell (namely PIC) method. The second one is a Lagrangian approach. To reduce this model (which is $6D$), while keeping the statistical character, we use a gyroaverage. A gyrokinetic theory [3] has been developed to reduce the problem to $5D$ unknowns.

Gyrokinetic model

Near the equilibrium, *i.e.* $\frac{B}{\delta B} \simeq \varepsilon$, with $\varepsilon \ll 1$, where B is the equilibrium magnetic field and δB is the perturbation. In this case, strong anisotropy appears in the tokamak. Let us take $\mathbf{k} = \mathbf{k}_{\parallel} + \mathbf{k}_{\perp}$ a wave vector, and define $\mathbf{b} = \frac{\mathbf{B}}{\|\mathbf{B}\|}$, we get

$$\frac{\|\mathbf{k}_{\parallel}\|}{\|\mathbf{k}_{\perp}\|} \simeq \varepsilon \quad (0.0.21)$$

Rather than following the fast rotational movement of the particle, we can use a gyroaverage and study the trajectory of the center guide. This means that we can reduce our model to $5D$ unknowns. Remark also, that the quantity :

$$\mu = m \frac{\|\mathbf{v}_{\perp}\|^2}{2\|\mathbf{B}\|} \quad (0.0.22)$$

becomes an adiabatic invariant, in the case of a non collisional model. So that the problem is rather $4D$ with a parameter μ . Now, let us go back to the Vlasov equation, and use the scaling of the perturbed equilibrium. We get the Vlasov scaled equation :

$$\partial_t f + v_{\parallel} \cdot \nabla_x f + \frac{1}{\varepsilon} v_{\perp} \cdot \nabla_x f - \varepsilon E_{\parallel} \cdot \nabla_v f + \left(E_{\perp} + \frac{1}{\varepsilon} v \times B \right) \cdot \nabla_v f = 0. \quad (0.0.23)$$

In the sequel, we consider a constant magnetic field parallel to the vector basis \mathbf{e}_3 , $\mathbf{B} = B\mathbf{e}_3$.

Following the change of variables proposed in [92]

$$\begin{aligned} x &\rightarrow R = x - \rho, \quad \text{where} \quad \rho = \frac{v_{\perp} \times \vec{e}_3}{B} \\ v &\rightarrow (\mu, \theta, v_{\parallel}), \quad \text{where} \quad \mu = \frac{v_{\perp}^2}{2B} \end{aligned}$$

which transforms $f(t, \mathbf{x}, \mathbf{v})$ into $f(t, \mathbf{R}, v_{\parallel}, \mu, \theta)$.

Finally, the Vlasov equation re-written in guiding center variables is

$$\begin{aligned} \partial_t f &+ \left(v_{\parallel} + \frac{\partial_R \Phi \times e_3}{B} \right) \cdot \partial_R f + \frac{B}{\varepsilon} \partial_{\theta} f - \partial_{\theta} \Phi \partial_{\mu} f \\ &- \varepsilon E \cdot \partial_{v_{\parallel}} f - \partial_R \Phi \cdot \frac{v \times e_3}{v_{\perp}^2} \partial_{\theta} f = 0. \end{aligned} \quad (0.0.24)$$

where Φ is the electric potential.

we can write a gyrokinetic model for $\langle f \rangle$ which is an approximation of the Vlasov model to the second order in ε

$$\partial_t \langle f \rangle + \left(v_{\parallel} + \frac{\partial_R \langle \Phi \rangle \times e_3}{B} \right) \cdot \partial_R \langle f \rangle - \langle E \rangle \partial_{v_{\parallel}} \langle f \rangle = \mathcal{O}(\varepsilon^2). \quad (0.0.25)$$

where $\langle \cdot \rangle$ is an average operator over θ .

Modern gyrokinetic models, are based on Hamiltonian dynamics. More details can be found in [3, 92, 94, 80, 34, 55].

Two-fluid Model

The two-fluid model is an intermediate model between Vlasov theory and MHD. It is a description of the plasma where the species ions/electrons form a system of mutually interacting. More details can be found in [71, 9].

MHD Model

The MHD model is the least detailed one. It is a description of the plasma as a single electrically conducting fluid. More details can be found in [71, 9, 29].

Presentation of this thesis

In this thesis, we tried to apply the IsoGeometric Analysis approach to solve some problems involved in Plasma Physics and Electromagnetism. As it is very common in Plasma Physics simulations, some numerical methods become in some sense references. We have applied the IGA approach, coupled with a Particle In Cell Method, to solve the Vlasov-Maxwell problem, in axisymmetric coordinates (*c.f* chapter 5). We have also developed a new Semi-Lagrangian method using mappings based on *B-splines*, to deal with complex geometries. This was done to solve the Vlasov-Poisson 2D problem (*c.f* chapter 6). We have applied the IGA method to solve the equivalent part of the Poisson's equation in the Gyrokinetic model, *i.e* Quasi-Neutral equation (*c.f* chapter 3). For the MHD equilibrium problem, we have constructed mappings that allow us to have an *aligned mesh* to the magnetic fields lines (*c.f* chapter 7). We have also studied and developed a new scheme to solve Maxwell's equations based on *B-splines*, and which allows us to remove one of the mass matrices; hence at each time step we will need to inverse only one matrix. We have also tried to develop a new DeRham sequence based on Box-splines (*c.f* chapter 8).

Remark 0.0.2 *It is very important to notice that one of the most important properties of IGA is that the domain description is given by the same basis functions used to approach the solution of the studied partial differential equations. This description stills exact after refinement; this explains the term iso-geometric. It turns out that in most applications (in plasma physics) we will consider, there will be no exact description of the domain, but just an approximation. In this case, it is rather an isoparametric method. The reader should pay attention to this particular point. We will use the isoparametric version of the IGA when physics requires. Otherwise, when the description is exact, we will use the conventional version of the IGA.*

At the beginning of each application chapter, we have tried to recall the physics that we are dealing with.

Publications

This thesis has lead to the following publications:

1. Arbitrary High-Order Spline Finite Element Solver for the Time Domain Maxwell equations (with E. Sonnendrücker). *Journal of Scientific Computing*, 2011, pages 1-20.

-
2. An Isogeometric Analysis Approach for the study of the gyrokinetic quasi-neutrality equation (with E. Sonnendrücker and N. Crouseilles). *Submitted*.
URL : <http://hal.inria.fr/inria-00587009>
 3. Solving the Vlasov equations in complex geometries. (with J. Abiteboul, G. Latu, V. Grandgirard, E. Sonnendrücker and A. Strugarek.). Proceedings of CEMRACS 2010. *Submitted*.
 4. An Axisymmetric PIC code based on Isogeometric Analysis (with A. Back, A. Crestetto, and E. Sonnendrücker). Proceedings of CEMRACS 2010. *Submitted*.

The following articles are in progress:

1. Simulation of 2D reduced MHD using Isogeometric Analysis (with E. Sonnendrücker). *In preparation*.
2. The Fast IGA Approach. *In preparation*.
3. *PyIGA*, Isogeometric Analysis simulations in *Python*. *In preparation*.
4. A new DeRham sequence based on Box-splines. *In preparation*.

Softwares

We finish this introduction with softwares developed during this thesis:

- **WEBSPLINE2D** is a Fortran code to solve 2D elliptic-pdes using the webspline method. (9'700 lines)
- **ISOBOX** is a Fortran code. This is the kernel of *PyIGA*. It contains a *C.A.D* module, that allows common geometric operations on NURBS curves. It contains a specific module for sparse matrices that arise in IGA. It also integrates a common tool for visualizations under VisIt, in *SILO* format. (15'000 lines)
- **ISOPIC** is a Fortran code, based on *ISOBOX*, that solves the coupled system Vlasov-Maxwell, in axisymmetric coordinates, using IGA. The particles are living in the patch. (8'900 lines)
- **PyIGA** is a Python library, based on *ISOBOX*, that allows the user to solve a large category of pdes (system of pdes), using the IsoGeometric Analysis approach. It also contains some *GB-splines*. (7'200 lines)
- **PyIGA - GUI** is a GUI that allows the user to create domains, refine and export them so they can be used by *PyIGA*. Data can be given in a file text or *XML* format. (2'900 lines)

Notations

- As it is a common use, we will denote a constant by C . If the constant depends on i parameters p_1, \dots, p_i , we will write $C = C(p_1, \dots, p_i)$
- $A \lesssim B$: if it exists a constant C such that $A \leq CB$.
- $A \simeq B$: if $A \lesssim B$ and $B \lesssim A$.
- **Multi-index** : $\mathbf{k} = (k_1, \dots, k_d)$
- D^i will denote the i^{th} derivative operator.
- $\mathcal{S}_k(T^*, \mathbf{m}, I)$, $\mathcal{S}_k(T, I)$ and \mathcal{S}_k will denote Schoenberg space. The latest notations are used, when parameters defining the Schoenberg space are fixed once for all.
- For each B-spline N_i , σ_i will denote its support.
- We will usually use the notation $b \in \Gamma$ or Λ , to describe the index of a univariate/multivariate *B-spline/NURBS*. In $1D$, we have $\Gamma = \{i/ 1 \leq i \leq N\}$. In $2D$, $\Gamma = \{(i_\xi, i_\eta)/ 1 \leq i_\xi \leq N^\xi, 1 \leq i_\eta \leq N^\eta\}$. When we are dealing with Dirichlet boundary condition, we will use Γ^0 or Λ^0 . They are obtained from Γ (or Λ) by removing interpolating *B-splines/NURBS*.
- $\mathcal{V}_h^0 = \text{span}\{\varphi_b, b \in \Lambda^0\}$ and $\mathcal{V}_h = \text{span}\{\varphi_b, b \in \Lambda\}$ will usually denote the finite dimensional space of $H_0^1(\Omega)$ and $H^1(\Omega)$.
- When we expand a function $u_h \in \mathcal{V}_h^0$ (or \mathcal{V}_h) over the basis of \mathcal{V}_h^0 (or \mathcal{V}_h), $[u_h]^b$ will denote the coefficients of the basis function φ_b .
- $\omega(g; h)$: **The modulus of continuity**, for a continuous function g , and a positif number h , we define: $\omega(g; h) := \max\{|g(x) - g(y)|; |x - y| \leq h\}$
- Ω : will denote a 2D physical (master) domain.
- \mathcal{P} : will denote a 2D patch (parametric) domain.
- Q : will usually denote a cell in the physical domain, (except in chapter 5, we will use C).
- \tilde{Q} : will usually denote a cell in the parametric domain, (except in chapter 5, we will use \tilde{C}).
- \mathcal{Q}_h : the set of all meshes in the parametric domain.
- \bar{Q} will denote the extension of the cell Q : i.e $\bar{Q} = \bigcup_{i \in \Gamma(Q)} \sigma_i$, where $\Gamma(Q) = \{i, \text{ such that } Q \cap \sigma_i \neq \emptyset\}$.

-
- $\mathcal{K}_h = \{\mathbf{F}(Q), Q \in \mathcal{Q}_h\}$: the set of all meshes in the physical domain.
 - \mathcal{N}_h : will denote the space of all *B-splines* functions according to a given set of meshes \mathcal{Q}_h .
 - \mathcal{V}_h : will denote the space of all *NURBS* according to a given set of meshes \mathcal{Q}_h .
 - The set of polynomials of degree less than k (\mathbf{k} , for multivariate polynomials), will be denote by $\mathcal{P}_{<k+1}$, \mathcal{P}_k , Π_k or $\Pi_{<k+1}$.
 - For variational formulations, we will usually index a multivariate basis function, by φ_b . The index b is intend to be a 2D index, *i.e* $b = (i,j)$.
 - π_X will usually denote either a L^2 Projector or Quasi-Interpolant on the space X .
 - $\|\cdot\|_{p,K}$ and $\|\cdot\|_{H^p(K)}$ will denote the norm of the Sobolev space $H^p(K)$.
 - $|\cdot|_{p,K}$ and $|\cdot|_{H^p(K)}$ will denote the semi-norm of the Sobolev space $H^p(K)$.

Contents

1	Splines and Isogeometric Analysis	1
1.1	Splines and B-splines functions	2
1.1.1	Splines	2
1.1.2	B-Splines	4
1.2	B-Spline series	5
1.2.1	Multivariate tensor product splines	8
1.3	Splines in CAD	9
1.3.1	Modeling a curve	9
1.3.2	Fundamental geometric operations	13
1.3.3	<i>NURBS</i>	15
1.3.4	Multivariate tensor product <i>NURBS</i>	15
1.3.5	Modeling conics using <i>NURBS</i>	16
1.4	Splines in Approximation Theory	17
1.4.1	Quasi-interpolant	18
1.4.2	Distance of a regular function to \mathcal{S}	18
1.5	Web-splines	19
1.5.1	B-Splines on bounded domains	20
1.5.2	Weight functions	20
1.5.3	Web-Splines	20
1.5.4	Stability in $w^e\mathbb{B}$	22
1.5.5	Study of an elliptic partial differential equation	22
1.5.6	Polynomial Approximation	24
1.5.7	Numerical results	24
1.5.8	Conclusion	26
1.6	Isogeometric analysis	26
1.6.1	Refinement strategies	28
1.6.2	Patch	28
1.6.3	Grid generation	29
1.6.4	Local approximation	29
1.6.5	Global approximation using <i>NURBS</i>	30
2	Elliptic Equations	31
2.1	Galerkin-Ritz approximation	32
2.2	The variational formulation	34
2.3	Assembling matrices	35
2.3.1	Stiffness local matrix	35
2.3.2	Mass local matrix	36
2.3.3	Local load vector	36

2.3.4	Assembling matrices algorithm	36
2.4	Numerical results	37
2.4.1	Domain defined by B-splines curves	37
2.4.2	Solution for an affine transformation	37
2.5	Domain defined with NURBS curves	40
2.5.1	Poisson's equation on a quarter ring domain	40
2.5.2	Poisson's equation on a ring domain	42
2.6	Computing the solution on general domain	42
2.7	Nonlinear elliptic problems	45
2.7.1	Picard's algorithm	47
2.7.2	Newton's algorithm	47
2.7.3	Numerical results : Example from combustion theory	48
3	Application to the Quasi-Neutral equation	51
3.1	Introduction	52
3.2	Quasi-neutrality equation	53
3.3	A fast solver for polar coordinates	54
3.4	Numerical validation	56
3.4.1	Test case 1: Order of convergence for Poisson in polar coordinates .	56
3.4.2	Test case 2: Chaotic solution	58
3.5	Numerical solution of the quasi-neutrality equation	59
3.5.1	The decoupling approach	60
3.5.2	First approach: spectral + finite differences	61
3.5.3	Second approach: FEM	62
3.5.4	Numerical results	62
3.6	Conclusion	65
4	Application to the 2D Maxwell's equations	67
4.1	Introduction	68
4.2	Variational formulation for the 2D Maxwell equations	69
4.3	Construction of the finite element spaces	70
4.3.1	Spline finite elements on patch grids	71
4.3.2	The Discrete Equations	72
4.4	Leap Frog scheme's stability	73
4.5	Numerical results	74
4.5.1	Test case 1: square	75
4.5.2	Test case 2: circular wave guide	77
4.5.3	Test case 3: Silver-Muller condition	78
4.6	H-rot formulation	79
4.7	Axisymmetric Variational formulation of the 2D Maxwell's equation	80
4.7.1	Discrete equations - 1 st formulation	81
4.7.2	Discrete equations - 2 nd formulation	83
4.7.3	H-rot formulation	84
4.7.4	Remarks	85
4.8	Conclusions and perspectives	86

5	An axisymmetric PIC code based on Isogeometric Analysis	97
5.1	Introduction	98
5.1.1	Domain parametrization using Splines/NURBS curves	98
5.2	PIC method for Vlasov equation	98
5.2.1	The PIC Method	99
5.2.2	The equations of motion	100
5.2.3	The Dirac mass with a change of variables	101
5.2.4	Computing J and ρ with a change of variables	102
5.3	Particles emission	102
5.3.1	Short description of a diode	102
5.3.2	Extraction conditions	102
5.4	Numerical results	103
5.4.1	Emission of particles in the diode	103
5.5	Conclusion and perspectives	103
6	Application to Semi-Lagrangian schemes	107
6.1	2D Vlasov	109
6.1.1	Physical model: the paraxial beam	109
6.1.2	The ISOLOSS code	110
6.2	Complex geometry using parametric surfaces	113
6.2.1	General framework	113
6.2.2	Analytic mapping	115
6.2.3	Bézier patches	115
6.3	Algorithms	116
6.3.1	Inverse mapping for Bézier patches	117
6.3.2	Reducing delays for patch finding	119
6.3.3	Velocity integrals	120
6.4	Results	120
6.4.1	Geometry settings and experimental results	120
6.4.2	Performance issues	122
7	Simulation of 2D reduced MHD	125
7.1	Introduction	126
7.2	Anisotropic Diffusion	126
7.2.1	Introduction	126
7.2.2	The choice of the grid	126
7.2.3	Evolution of a Gaussian pulse	127
7.2.4	Conclusions	128
7.3	MHD equilibrium	130
7.3.1	Equilibrium in the absence of toroidal flux	130
7.3.2	Equilibrium with toroidal flux	132
7.3.3	Nonlinear equilibrium	133
7.4	Current-Hole	134
7.4.1	Time scheme	135
7.4.2	Variational formulation	135
7.4.3	Numerical results	137
7.5	Conclusions	144

8	A new DeRham sequence based on Box-splines	145
8.1	Introduction	146
8.2	Notations	146
8.3	Bernstein-Bézier bivariate polynomials	147
8.4	Box-Splines	148
8.4.1	Strang-Fix conditions applied to Box-splines	150
8.4.2	Box-Spline series	151
8.4.3	Quasi-interpolant operator for Box-Splines	153
8.5	Box-splines as finite elements basis	153
8.5.1	Approximation with box-splines	154
8.6	DeRham diagram	155
8.6.1	Notations	156
8.6.2	Interpolants and commutativity	156
8.6.3	Approximation Analysis	158
8.7	Boundary Condition using Box-splines	159
8.8	Conclusions and Perspectives	159
A	Appendix: Decoupling approach for $J(r, \theta)$	163
B	Transformation compatible with grad, div and curl operators	165
C	Equations of motion	167
C.1	Cylindrical coordinates	167
C.2	General coordinates	171
C.3	Numerical implementation	176
C.4	Computing densities ρ and \mathbf{J}	177
D	Python Interface	181
D.1	Introduction	182
D.2	pdefield class	182
D.3	source class	182
D.4	isogeo class	182
D.4.1	Setting Grid's parameters	184
D.4.2	Poisson's equation	185
D.4.3	Anisotropic Diffusion	190
D.4.4	Maxwell's 2D problem	193
D.5	Using GB-splines	206
D.6	PyIGA's available operators	206
D.7	PyIGA's input data	208
D.7.1	Importing domain data	208
D.7.2	Refinement data	209
D.7.3	Boundary conditions	210
D.7.4	Silver Muller condition	211
D.7.5	Details and output file	211
D.8	Creating domains using the GUI	211
D.8.1	The XML Format	211
D.8.2	Using the GUI	211
D.8.3	Examples	212
D.9	Creating domains defined by an implicit function	213

Contents

D.10 Visualization	215
D.10.1 Using Silo	215
D.10.2 Using VTK	216
D.11 PyIGA and Pastix	216
D.12 Installing PyIGA	216
Bibliography	219

Splines and Isogeometric Analysis

Contents

1.1 Splines and B-splines functions	2
1.1.1 Splines	2
1.1.2 B-Splines	4
1.2 B-Spline series	5
1.2.1 Multivariate tensor product splines	8
1.3 Splines in CAD	9
1.3.1 Modeling a curve	9
1.3.2 Fundamental geometric operations	13
1.3.3 <i>NURBS</i>	15
1.3.4 Multivariate tensor product <i>NURBS</i>	15
1.3.5 Modeling conics using <i>NURBS</i>	16
1.4 Splines in Approximation Theory	17
1.4.1 Quasi-interpolant	18
1.4.2 Distance of a regular function to \mathcal{S}	18
1.5 Web-splines	19
1.5.1 B-Splines on bounded domains	20
1.5.2 Weight functions	20
1.5.3 Web-Splines	20
1.5.4 Stability in $w^e\mathbb{B}$	22
1.5.5 Study of an elliptic partial differential equation	22
1.5.6 Polynomial Approximation	24
1.5.7 Numerical results	24
1.5.8 Conclusion	26
1.6 Isogeometric analysis	26
1.6.1 Refinement strategies	28
1.6.2 Patch	28
1.6.3 Grid generation	29
1.6.4 Local approximation	29
1.6.5 Global approximation using <i>NURBS</i>	30

1.1 Splines and B-splines functions

1.1.1 Splines

Splines are piecewise polynomials defined on the real line. We shall require that on each compact interval, they consist of a small number of non-vanishing polynomial pieces.

Let $T^* = \{t_i^*, 0 \leq i \leq s\}$ be a finite strictly increasing sequence of points of \mathbb{R} . A function S on \mathbb{R} is a spline of order $k, k \geq 1$ with the breakpoints T^* if on each interval $(t_i^*, t_{i+1}^*),$ it is a polynomial of degree $\leq p := k - 1$. On the other hand, the spline can have any regularity less than $p - 1$ at the breakpoints. The smoothness r_i of a spline S at the breakpoint t_i^* is defined as follows:

- $r_i := 0$ if S is discontinuous at t_i^* , otherwise
- r_i is the largest integer $0 < r_i \leq k$ so that S has continuous derivatives of orders $< r_i$

therefore, we denote the associated spline space on an interval $I = [a, b], a := t_0^*, b = t_s^*$ by $S_k^*(T^*, I)$ which consists of all splines S of order $\leq k$ with breakpoints contained in T^* and of smoothness $\geq r_i$ at t_i^* .

Rather than use the smoothness of the spline at breakpoints, we use the defect $m_i := k - r_i$, this is the number of degrees of freedom of S at t_i^* .

A simple computation leads to $\dim S_k(T^*, \mathbf{m}, I) = k + \sum_{i=0}^s m_i$ with $\mathbf{m} := (m_0, \dots, m_s)$. The space $S_k(T^*, \mathbf{m}, I)$ is called the Schoenberg space.

In fact, a natural basis of the Schoenberg space is

$$S_{-j}(x) = \frac{(x-a)^j}{j!}, \quad j = 0, \dots, k$$

$$S_{i,j}(x) = \frac{(x-t_i^*)_+^j}{j!}, \quad j = k - m_i, \dots, k - 1, \quad i = 1, \dots, s$$

Then, for each spline $S \in S_k(T^*, \mathbf{m}, I)$, we can write

$$S = \sum_{j=0}^{k-1} a_{-j}(S) S_{-j}(x) + \sum_{i=1}^s \sum_{j=k-m_i}^{k-1} a_{i,j}(S) S_{i,j}(x) \quad (1.1.1)$$

where $a_{-j}(S) = S^{(j)}(a), \forall j \in \{0, \dots, k-1\}$, with $S^{(j)}(a)$ the j^{th} derivative of S evaluated at a , and $a_{i,j}(S) = S^{(j)}(t_i^*+) - S^{(j)}(t_i^*-), \forall i \in \{1, \dots, s\}, \forall j \in \{k - m_i, \dots, k - 1\}$. This introduces dual functionals for the basis. This also, gives us the dimension of the Schoenberg space $S_k(T^*, \mathbf{m}, I)$, which writes:

$$\dim S_k(T^*, \mathbf{m}, I) = n + k, \quad n = \sum_{i=1}^s m_i. \quad (1.1.2)$$

One of the basic properties used in the finite element method, is the Markov inequalities for polynomials. Splines also, verify such elementary inequalities:

Theorem 1.1.1 *If the breakpoints T^* satisfy $\delta_0 \leq |t_{j+1}^* - t_j^*| \leq \delta, \quad j \in \{0, \dots, s\}$, then for each $S \in S_k(T^*, \mathbf{m}, I)$, we have*

$$\|S\|_p \leq C \delta^{\frac{1}{q} - \frac{1}{p}} \|S\|_q, \quad p_0 \leq q \leq p \leq +\infty$$

1.1. Splines and B-splines functions

where $C = C(p_0, k)$ is a constant. We also have

$$\|S^{(j)}\|_p \leq C\delta_0^{-j}\|S\|_p, \quad 0 \leq j \leq +\infty, j \in \{1, \dots, k-1\}$$

Expanding splines as in (1.1.1) would be a very difficult task. Thus Curry and Schoenberg [26] have introduced another basis which has a more local character. Their basis are splines with the smallest possible support. They are defined by means of the divided differences and called basic splines (B-splines). For more details on this subject we refer to the books of De-Boor [19] (for computational aspect), DeVore and Lorentz [33], and Schumaker [98] (for more theoretical aspects).

Definition 1.1.2 (Divided Differences) For a set of points (not necessarily ordered) $X := \{x_0, \dots, x_n\}$, and a function f , we define the n -th divided difference of f by

$$[x_0, \dots, x_n]f := A_n \tag{1.1.3}$$

where A_n is the coefficient of x^n of the polynomial which interpolates f at x_0, \dots, x_n .

For example, $[x_0]f = f(x_0)$

$$[x_0, x_1]f = \frac{f(x_0) - f(x_1)}{x_0 - x_1}, \quad x_0 \neq x_1, \quad \text{otherwise } [x_0, x_1]f = f'(x_0)$$

We present here some properties of the Divided Differences operator:

- $[x_0, \dots, x_n]f$ is a linear combination of the derivatives $f^{(l)}(x_i)$, $0 \leq l \leq m_i - 1$, where m_i is the multiplicity of the point x_i in the set X ,
- $[x_0, \dots, x_n]f$ is symmetric in x_0, \dots, x_n ,
- $[x_0, \dots, x_n]f$ is constant if f is a polynomial of degree $\leq n$, and zero for a polynomial of degree $< n$,
- $[x_0, \dots, x_0]f = \frac{1}{n!}f^{(n)}(x_0)$
- (Newton's Formula) $P_n(f, X; x) = \sum_{i=0}^n \prod_{j=0}^{i-1} (x - x_j)[x_0, \dots, x_i]f$ is the interpolating polynomial for f at the sites X .
- if $f \in \mathcal{C}^n([a, b])$, $a \leq x_i \leq b$, $0 \leq i \leq n$, then :

$$[x_0, \dots, x_n]f = \frac{1}{n!}f^{(n)}(\xi), \quad \text{for some } \xi \in [a, b]$$

- $[x_0, \dots, x_n]f$ is continuous at the sites in X , if the derivatives of f of proper orders are continuous at the considered site,
- if $x_0 \neq x_n$, we have

$$[x_0, \dots, x_n]f = \frac{1}{x_n - x_0} \{ [x_1, \dots, x_n]f - [x_0, \dots, x_{n-1}]f \} \tag{1.1.4}$$

- (Leibniz's Formula) $[x_0, \dots, x_n](fg) = \sum_{i=0}^n [x_0, \dots, x_i](f) [x_i, \dots, x_n](g)$

1.1. Splines and B-splines functions

- if $f^{(n-1)}$ is absolutely continuous, and if not all x_i coincide, we have

$$[x_0, \dots, x_n]f = \int_0^1 dt_1 \int_0^{t_1} dt_2 \cdots \int_0^{t_{n-1}} f^{(n)}(x_0 + h_1 t_1 + h_2 t_2 + \cdots + h_n t_n) dt_n$$

where we denote $h_i = x_{i+1} - x_i$, $i \in \{0, \dots, n-1\}$. This formula is very important; in fact, later, we will define another family of splines, using a generalization of this formula. Another application of this formula, is the next result, where under the same assumptions we have:

$$|[x_0, \dots, x_n]f| \leq \frac{1}{n!} \|f^{(n)}\|_\infty$$

This shows, that the functional $f \rightarrow [x_0, \dots, x_n]f$ is continuous on $C^n[a, b]$. To finish, we can give a representation of the functional $[x_0, \dots, x_n]$ in term of the Peano kernel:

$$[x_0, \dots, x_n]f = \int_a^b f^{(n)}(t)[x_0, \dots, x_n] \left(\frac{(\cdot - t)_+^{n-1}}{(n-1)!} \right) dt$$

this result is valid under the assumption that all x_i are distinct.

1.1.2 B-Splines

Definition 1.1.3 (B-Spline) Let $X = \{x_0, \dots, x_p\}$ a non-decreasing sequence of $p+1$ points such that $x_0 \neq x_p$. The B-Spline will be defined in term of the following Divided-Difference :

$$M(x) = M(x; X) = M(x; x_0, \dots, x_p) = p[x_0, \dots, x_p](\cdot - x)_+^{p-1} \quad (1.1.5)$$

The points forming the set X , are called *knots*, and X is said to be a *knot vector*. From the properties of the Divided-Differences, we can easily prove the following results:

- $M(x) = 0$, if $x < x_0$ or $x_p < x$
- $M(x) > 0, \forall x \in [x_0, x_p]$
- $\frac{M}{n!}$ is the Peano kernel of the divided-difference at the set $X = \{x_0, \dots, x_p\}$. Then, for any $f \in W_1^p$, we have,

$$[x_0, \dots, x_p]f = \int_{-\infty}^{+\infty} f^{(p)}(t)M(t)dt$$

- $\int_{-\infty}^{+\infty} M(t)dt = 1$
- $M(x) \sim (x - x_0)^{p-m_0}$, $x \rightarrow x_0+$
- $M(x) \sim (x - x_p)^{p-m_p}$, $x \rightarrow x_p-$
- $M(x) \leq \frac{C}{x_p - x_0}$, where $C = C(p)$ is a constant depending only on p
- if $p \geq 2$, we have the following recurrence formula, which will serve to define a simpler form for B-splines,

$$M(x; x_0, \dots, x_p) = \frac{p}{p-1} \left\{ \frac{x - x_0}{x_p - x_0} M(x; x_0, \dots, x_{p-1}) + \frac{x_p - x}{x_p - x_0} M(x; x_1, \dots, x_p) \right\} \quad (1.1.6)$$

1.2. B-Spline series

In fact, we can define the *B-spline*, depending on a normalization condition. The previous one, was done to have integral one.

Another normalization is to define the *B-spline* as:

$$N(x; x_0, \dots, x_p) = \frac{1}{p}(x_p - x_0)M(x; x_0, \dots, x_p) \quad (1.1.7)$$

which will lead to the partition unity property.

Hence, the recurrence formula (1.1.6) gives :

$$N(x; x_0, \dots, x_p) = \frac{x - x_0}{x_{p-1} - x_0}N(x; x_0, \dots, x_{p-1}) + \frac{x_p - x}{x_p - x_1}N(x; x_1, \dots, x_p) \quad (1.1.8)$$

What is interesting in this formula, is that in the right hand side, each term depends only on the B-spline knots, and not on its degree.

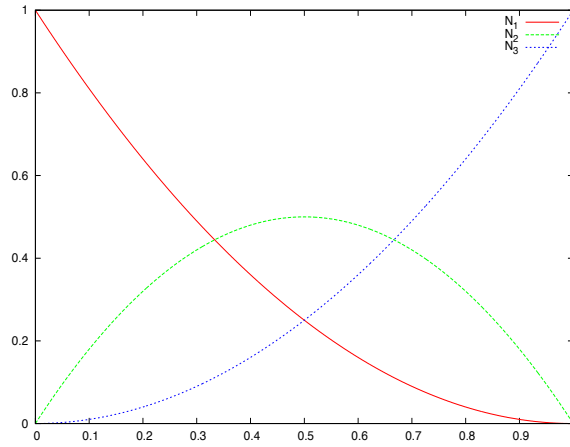


Figure 1.1: B-splines functions associated to the knot vector $T = \{000\ 111\}$, of order $k = 3$. These are Bernstein polynomials

1.2 B-Spline series

As noticed in the previous section, to construct a *B-spline* of degree p , we need $p+1$ knots. Then, to create a family of *B-splines*, we will need to have a non-decreasing sequence of knots, also called knot vector.

Let $T = (t_i)_{1 \leq i \leq N+k}$ be a non-decreasing sequence of knots, with $k = p + 1$. Each set of knots $T_j = \{t_j, \dots, t_{j+p}\}$ will generate a *B-spline* N_j . This leads to the following definition:

Definition 1.2.1 (B-Spline serie) The j -th *B-Spline* of order k is defined by the recurrence relation:

$$N_j^k = w_j^k N_j^{k-1} + (1 - w_{j+1}^k) N_{j+1}^{k-1}$$

where,

$$w_j^k(x) = \frac{x - t_j}{t_{j+k-1} - t_j} \quad N_j^1(x) = \chi_{[t_j, t_{j+1}[}(x)$$

for $k \geq 1$ and $1 \leq j \leq N$.

1.2. B-Spline series

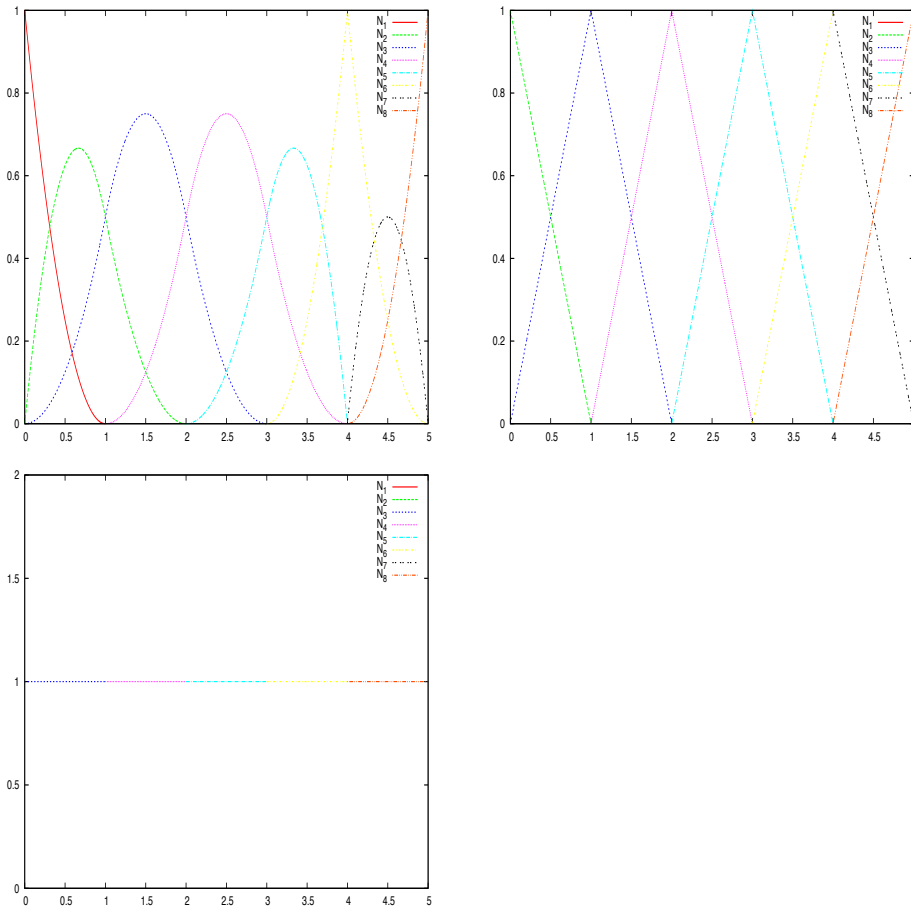


Figure 1.2: B-splines functions associated to the knot vector $T = \{000\ 1\ 2\ 3\ 44\ 555\}$, of order $k = 1, 2, 3$

Curry and Schoenberg proved that

Theorem 1.2.2 *With the above notations, the B-splines $(N_j)_{j \in \Gamma}$ are basis for the space $\mathcal{S}_k = \mathcal{S}_k(T, I)$.*

We note some important properties of a B-splines basis:

- B-splines are piecewise polynomial of degree $p = k - 1$
- Compact support; the support of N_j^k is contained in $[t_j, \dots, t_{j+k}]$
- if $x \in]t_j, t_{j+1}[$, then only the B-splines $\{N_{j-k+1}^k, \dots, N_j^k\}$ are non vanishing at x
- Positivity: $\forall j \in \{1, \dots, N\} \ N_j(x) > 0, \ \forall x \in]t_j, \dots, t_{j+k}[$
- Partition of unity : $\sum_{i=1}^N N_i^k(x) = 1, \forall x \in \mathbb{R}$
- Local linear independence
- If a knot t_i has a multiplicity m_i then the B-spline is $\mathcal{C}^{(p-m_i)}$ at t_i

We now give some important properties of the B-splines series:

1.2. B-Spline series

Marsden's identity

We have the following result:

$$(\cdot - u)^{k-1} = \sum_j \psi_{j,k}(u) N_j^k(\cdot) \quad (1.2.9)$$

where $\psi_j^k(u) = \prod_{i=1}^{k-1} (t_{j+i} - u)$. This property is very important, it says that we can reproduce any polynomial of degree less than p , on a *B-spline* basis, which is very crucial in a finite element method.

This leads to the following representation theorem,

Theorem 1.2.3 *Expansion in B-splines form: if $S \in S_k(T, I)$ and $x \in [t_j, t_{j+1}]$, then we can write S as :*

$$S(x) = \sum_{i=j-k+1}^j [S]^i N_i^k(x)$$

Convex hull

If $t_j < x < t_{j+1}$, then $S(x) = \sum_{i=j-k+1}^j [S]^i N_i^k(x)$ is a strictly convex combination of the numbers $([S]^i)_{i=j-k+1, \dots, j}$.

Using the convex-hull property, if $t_j \leq x \leq t_{j+1}$, then one can easily check that

$$\min([S]^i)_{i=j-k+1, \dots, j} \leq S(x) \leq \max([S]^i)_{i=j-k+1, \dots, j} \quad (1.2.10)$$

Conditioning property (stability)

Let $\alpha = \{\alpha_i, i \in \Lambda\}$, be a sequence of reals. We have the following conditioning property:

$$|\alpha_j| \leq C_{k, \infty} \left\| \sum_{i \in \Lambda} \alpha_i N_i^k \right\|, \quad \forall j \in \Lambda \quad (1.2.11)$$

where $C_{k, \infty} \leq \frac{k}{2^{k-1}}$ (Scherer and Shadrin).

Therefore, we have

$$\frac{1}{C_{k, \infty}} \|\alpha\| \leq \left\| \sum_{i \in \Lambda} \alpha_i N_i^k \right\| \leq \|\alpha\| := \max_{i \in \Lambda} (|\alpha_i|) \quad (1.2.12)$$

Dual functionals

For each $f \in S_k(T, I)$, we define dual basis functionals of the *B-splines* :

$$\lambda_j f = \sum_{\mu=1}^k \frac{(-D)^{k-\mu} \psi_j(\tau_j)}{(k-1)!} D^{\mu-1} f(\tau_j) \quad (1.2.13)$$

with τ_j is taken arbitrarily in $]t_j, t_{j+k}[$. Here D^i denotes the i^{th} derivative operator.

1.2. B-Spline series

Variation Diminution, Schoenberg (1967)

Let $S^- f$ be the number of sign changes of the function $f(x) = \sum_i [f]^i N_i^k(x)$.

If $S^- \alpha$ computes the number of changing sign of a real valued finite sequence $\alpha = (\alpha_i)_{1 \leq i \leq N}$, we have,

$$S^- \left(\sum_j \alpha_j N_j \right) \leq S^- \alpha \quad (1.2.14)$$

In the section 1.3.2, we will show how one can insert new knots into the vector, and the impact of this operation on the control points. We give here another impact in term of the variation diminution :

Let $f, (\tilde{f})$ be a spline associated to the knot vector $T, (\tilde{T})$, where \tilde{T} is obtained from T by inserting a new knot. Then we have,

$$S^- \tilde{f} \leq S^- f \quad (1.2.15)$$

Deriving a B-spline

For a general derivative of a *B-spline* we have the following result:

$$N_i^{k(l)} = \frac{p!}{(p-l)!} \sum_{j=0}^l a_{l,j} N_{i+j}^{k-l}, \quad l \leq k-1 \quad (1.2.16)$$

where the coefficient $a_{l,j}$ are given :

$$a_{0,0} = 0, \quad a_{l,j} = \frac{a_{l-1,j} - a_{l-1,j-1}}{t_{i+j+k-l} - t_{i+j}}, \quad \text{for each } j \in \{0, \dots, l-1\}$$

$$a_{l,0} = \frac{a_{l-1,0}}{t_{i+k-l} - t_i}, \quad a_{l,l} = \frac{-a_{l-1,l-1}}{t_{i+k} - t_{i+l}}$$

1.2.1 Multivariate tensor product splines

Let us consider d knot vectors $\mathcal{T} = \{T^1, T^2, \dots, T^d\}$. For simplicity, we consider that those knot vectors are open, and of bounds 0 and 1. In the sequel we will use the notation $I = [0, 1]$. Each knot vector T^i , will generate a basis for a Schoenberg space, $\mathcal{S}_{k_i}(T^i, I)$. The tensor product of all those spaces is also a Schoenberg space, namely $\mathcal{S}_{\mathbf{k}}(\mathcal{T})$, where $\mathbf{k} = \{k_1, \dots, k_d\}$. The cube $\mathcal{P} = I^d = [0, 1]^d$, will be referred to as a patch.

The basis for $\mathcal{S}_{\mathbf{k}}(\mathcal{T})$ is defined by a tensor product :

$$N_{\mathbf{i}}^{\mathbf{k}} := N_{i_1}^{k_1} \otimes N_{i_2}^{k_2} \otimes \dots \otimes N_{i_d}^{k_d}$$

where, $\mathbf{i} = \{i_1, \dots, i_d\}$.

A typical cell from \mathcal{P} is a cube of the form : $Q_{\mathbf{i}} = [\xi_{i_1}, \xi_{i_1+1}] \otimes \dots \otimes [\xi_{i_d}, \xi_{i_d+1}]$. To any cell Q , we will associate its extension \tilde{Q} , which is the union of the supports of basis functions, that intersects Q .

All results presented in the previous sections, can be easily extended to multivariate splines.

1.3 Splines in CAD

1.3.1 Modeling a curve

In the sequel, we will explain how we can model curves, beginning from the simplest one (the p-form) to *B-splines*, *NURBS* curves. We will explain the advantages of each method. Most of the results presented in this section were taken from [91]. We will denote by $\mathcal{C}(\mathbf{x}(t))$ a parametric curve, where each point is defined by the value of the parameter t .

The p-form

The first way to model a curve would be to consider the following form :

$$\mathcal{C}(\mathbf{x}(t)) = \sum_{i=0}^n t^i \mathbf{P}_i \quad (1.3.17)$$

Let us denote $\mathbf{x}(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}$, and $\mathbf{P}_i = \begin{pmatrix} P_i^x \\ P_i^y \\ P_i^z \end{pmatrix}$. The relation (1.3.17) can be written,

$$\begin{cases} x(t) = \sum_{i=0}^n t^i P_i^x \\ y(t) = \sum_{i=0}^n t^i P_i^y \\ z(t) = \sum_{i=0}^n t^i P_i^z \end{cases} \quad (1.3.18)$$

A simple computation leads to $\mathbf{P}_i = \frac{1}{i!} \frac{d^{(i)}}{dt} \mathcal{C}(\mathbf{x}(t))|_{t=0}$, for each $i \in \{0, \dots, n\}$. Even if the p-form is a natural description for curves, it presents some disadvantages :

- the curve is not necessary regular everywhere. So a regular description of the curve, may lead to non-efficient approximation,
- the points $(\mathbf{P}_i)_{0 \leq i \leq n}$ do not have any geometric interpretation,
- numerical evaluation of such description, needs the use of Horner algorithm, which is unstable.

The Bézier-form

Rather than use $\{1, t, \dots, t^n\}$ as a basis of $\Pi_{<n+1}$, we can take Bernstein polynomials; this leads to the Bézier-form. Therefore, it is equivalent to the p-form and writes :

$$\mathcal{C}(\mathbf{x}(t)) = \sum_{i=0}^n B_i^n(t) \mathbf{P}_i, \quad \text{for each } 0 \leq t \leq 1 \quad (1.3.19)$$

where B_i^n denote Bernstein polynomials :

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i}, \quad \text{for each } 0 \leq t \leq 1 \quad (1.3.20)$$

The sequence $(\mathbf{P}_i)_{0 \leq i \leq n}$ is called control points.

Examples

1.3. Splines in CAD

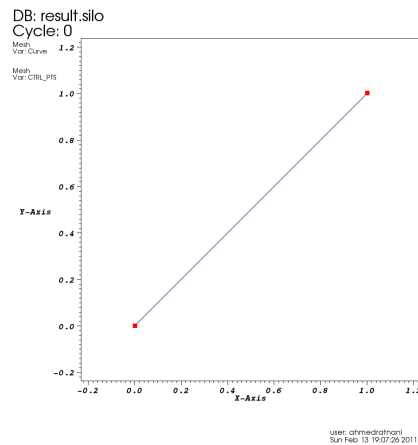


Figure 1.3: A Bézier-curve of degree 1 and its control points

- $n = 1$:

$$\mathcal{C}(\mathbf{x}(t)) = (1 - t)\mathbf{P}_0 + t\mathbf{P}_1, \text{ for each } 0 \leq t \leq 1.$$

This is just a description of a segment (figure 1.3).

- $n = 2$:

$$\mathcal{C}(\mathbf{x}(t)) = (1 - t)^2\mathbf{P}_0 + 2t(1 - t)\mathbf{P}_1 + t^2\mathbf{P}_2, \text{ for each } 0 \leq t \leq 1.$$

This forms a parabolic arc (figure 1.4). We have the following properties:

- $\{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2\}$ form a polygon: this is the control polygon,
- $\mathbf{P}_0 = \mathcal{C}(\mathbf{x}(0))$, and $\mathbf{P}_2 = \mathcal{C}(\mathbf{x}(1))$,
- $\mathcal{C}'(\mathbf{x}(0))$ is parallel to $\mathbf{P}_1 - \mathbf{P}_0$, and $\mathcal{C}'(\mathbf{x}(1))$ is parallel to $\mathbf{P}_2 - \mathbf{P}_1$,
- the curve is contained in the triangle $\mathbf{P}_0\mathbf{P}_1\mathbf{P}_2$,
- the control points *approach the behavior* of the curve.

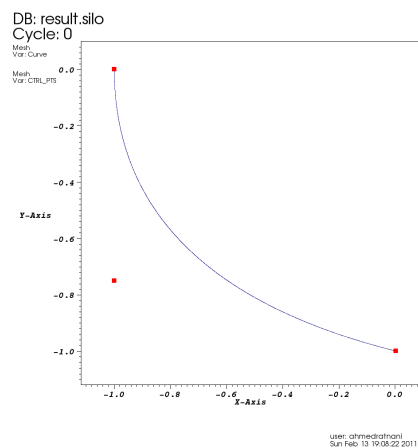


Figure 1.4: A Bézier-curve of degree 2 and its control points

1.3. Splines in CAD

- $n = 3$:

$\mathcal{C}(\mathbf{x}(t)) = (1-t)^3\mathbf{P}_0 + 3t^2(1-t)\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_3$, for each $0 \leq t \leq 1$. We give an example of such a curve (figure 1.5)

We have the following properties:

- $\{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3\}$ form a polygon: this is the control polygon,
- $\mathbf{P}_0 = \mathcal{C}(\mathbf{x}(0))$, and $\mathbf{P}_3 = \mathcal{C}(\mathbf{x}(1))$,
- $\mathcal{C}'(\mathbf{x}(0))$ is parallel to $\mathbf{P}_1 - \mathbf{P}_0$, and $\mathcal{C}'(\mathbf{x}(1))$ is parallel to $\mathbf{P}_3 - \mathbf{P}_2$,
- the control points *approach the behavior* of the curve.
- convex-hull : the curve is contained in the convex-hull associated to the control points,
- variation diminution : there is no line that intersects the control polygon in more points than the curve,
- it associates an implicit direction to the curve.

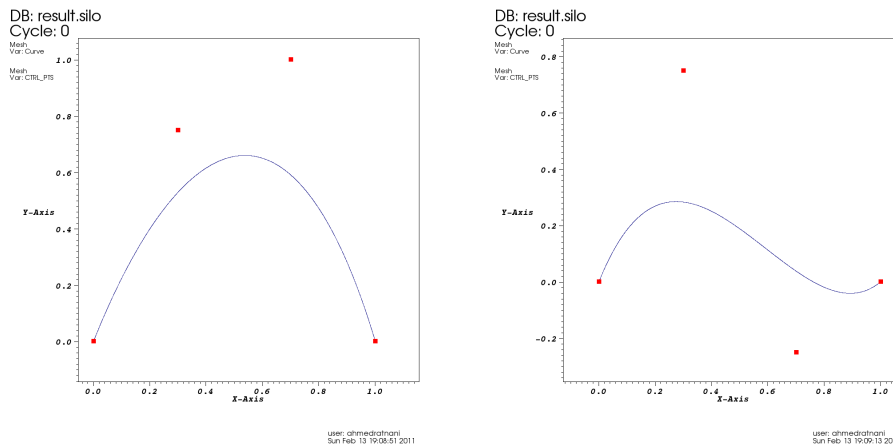


Figure 1.5: A Bézier-curve of degree 3 and its control points. (right) The after moving one control point.

Bézier-curves properties

We summarize the properties revealed in the previous examples:

- Invariance under some transformations : rotation, translation, scaling; it is sufficient to transform the control points,
- $B_i^n(t) \geq 0, \forall 0 \leq t \leq 1$
- partition of unity : $\sum_{i=0}^n B_i^n(t) = 1, \forall 0 \leq t \leq 1$
- $B_0^n(0) = B_n^n(1) = 1$
- each B_i^n has exactly one maximum in $[0, 1]$, at $\frac{i}{n}$
- B_i^n are symmetric with respect to $\frac{1}{2}$
- recursive property : $B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t)$, and $B_i^n(t) = 0$, if $i < 0$ or, $i > n$

1.3. Splines in CAD

- derivation property: $B_i^{n'}(t) = n\{B_{i-1}^{n-1}(t) - B_i^{n-1}(t)\}$, and $B_{-1}^{n-1}(t) \equiv B_n^{n-1}(t) \equiv 0$
- deriving a curve : $C'(t) = n\{\sum_{i=0}^{n-1} B_i^{n-1}(t) (\mathbf{P}_{i+1} - \mathbf{P}_i)\}$
hence, we have :

$$C'(0) = n(\mathbf{P}_1 - \mathbf{P}_0) \quad C'(1) = n(\mathbf{P}_n - \mathbf{P}_{n-1}) \quad (1.3.21)$$

and,

$$C''(0) = n(n-1)(\mathbf{P}_0 - 2\mathbf{P}_1 + \mathbf{P}_2) \quad C''(1) = n(\mathbf{P}_n - 2\mathbf{P}_{n-1} + \mathbf{P}_{n-2}) \quad (1.3.22)$$

this shows the interest of Bézier curves.

- DeCasteljau algorithm:

$$C^n(t; \mathbf{P}_0, \dots, \mathbf{P}_n) = (1-t)C^{n-1}(t; \mathbf{P}_0, \dots, \mathbf{P}_{n-1}) + tC^{n-1}(t; \mathbf{P}_1, \dots, \mathbf{P}_n) \quad (1.3.23)$$

The spline-form

Even with the advantages of the Bézier-form, still the problem of the regularity of the curve. We need to use a piecewise-polynomial form. For this purpose, we use *B-splines*. As seen before, they form a basis for the Schoenberg space.

Let $(\mathbf{P}_i)_{1 \leq i \leq N} \in \mathbb{R}^d$ be a sequence of control points, forming a control polygon.

Definition 1.3.1 (B-Spline curve) The *B-spline curve* in \mathbb{R}^d associated to $T = (t_i)_{1 \leq i \leq N+k}$ and $(\mathbf{P}_i)_{1 \leq i \leq N}$ is defined by :

$$C(t) = \sum_{i=1}^N N_i^k(t) \mathbf{P}_i$$

We have the following properties for a *B-spline curve*:

- If $N = k$, then C is just a Bézier-curve,
- C is a piecewise polynomial curve,
- The curve interpolates its extremas if the associated multiplicity of the first and the last knot are maximum (*i.e.* equal to k),
- Invariance with respect to affine transformations,
- strong convex-hull property:
if $t_i \leq t \leq t_{i+1}$, then $C(t)$ is inside the convex-hull associated to the control points $\mathbf{P}_{i-p}, \dots, \mathbf{P}_i$,
- local modification : moving \mathbf{P}_i affects $C(t)$, only in the interval $[t_i, t_{i+k}]$,
- the control polygon approaches the behavior of the curve

Remark 1.3.2 we can use multiple control points : $\mathbf{P}_i = \mathbf{P}_{i+1}$.

1.3. Splines in CAD

Deriving a B-spline curve: In the previous section, we gave a formula to compute B-spline's derivatives. Let us see what happens for C' :

$$C'(t) = \sum_{i=1}^n \left(N_i^{k'}(t) \mathbf{P}_i = \sum_{i=1}^n \frac{p}{t_{i+p} - t_i} N_i^{k-1}(t) \mathbf{P}_i - \frac{p}{t_{i+1+p} - t_{i+1}} N_{i+1}^{k-1}(t) \mathbf{P}_i \right) = \sum_{i=1}^{n-1} N_i^{k-1*}(t) \mathbf{Q}_i \quad (1.3.24)$$

where $\mathbf{Q}_i = p \frac{\mathbf{P}_{i+1} - \mathbf{P}_i}{t_{i+1+p} - t_{i+1}}$, and $\{N_i^{k-1*}, 1 \leq i \leq n-1\}$ are generated using the knot vector $T^* = \{0 \cdots 0, \dots, 1 \cdots 1\}$, where we have reduced by one the multiplicity of the first and the last knot (in the case of opened knot vector).

example: $T = \{000 \frac{2}{5} \frac{3}{5} 111\}$, $p = 2, n = 5$.

We have $C(t) = \sum_{i=1}^5 N_i^{3'}(t) \mathbf{P}_i$, then

$$C'(t) = \sum_{i=1}^4 N_i^{2*}(t) \mathbf{Q}_i$$

where

$$\mathbf{Q}_1 = 5\{\mathbf{P}_2 - \mathbf{P}_1\} \quad \mathbf{Q}_2 = \frac{10}{3}\{\mathbf{P}_3 - \mathbf{P}_2\}$$

$$\mathbf{Q}_3 = \frac{10}{3}\{\mathbf{P}_4 - \mathbf{P}_3\} \quad \mathbf{Q}_4 = 5\{\mathbf{P}_5 - \mathbf{P}_4\}$$

the B-splines $\{N_i^{2*}, 1 \leq i \leq 4\}$ are associated to the knot vector $T^* = \{00 \frac{2}{5} \frac{3}{5} 11\}$.

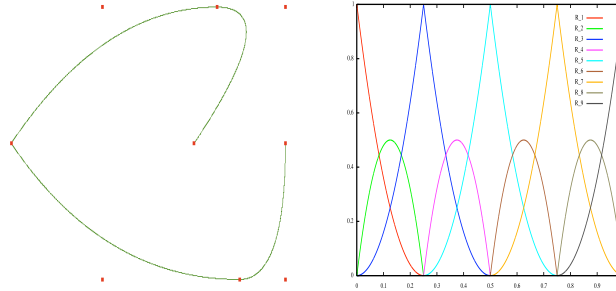


Figure 1.6: (left) A B-spline curve and its control points, (right) B-splines functions used to draw the curve. $N = 9, p = 2, T = \{000, \frac{1}{4} \frac{1}{4}, \frac{1}{2} \frac{1}{2}, \frac{3}{4} \frac{3}{4}, 111\}$

Definition 1.3.3 (B-spline surface) The B-spline surface of order k associated to the knot vectors $\{T^{(1)}, T^{(2)}\}$, the control points $(\mathbf{P}_{i,j})_{1 \leq i \leq N_1, 1 \leq j \leq N_2}$, is defined by

$$\mathbf{M}(t^{(1)}, t^{(2)}) = \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} N_{i,j}(t^{(1)}, t^{(2)}) \mathbf{P}_{i,j}$$

with $N_{i,j}(t^{(1)}, t^{(2)}) = N_i^{(1)}(t^{(1)}) N_j^{(2)}(t^{(2)})$

1.3.2 Fundamental geometric operations

After modification, we denote by $\tilde{N}, \tilde{k}, \tilde{T}$ the new parameters. (\mathbf{Q}_i) are the new control points.

Knot insertion

One can insert a new knot t , where $t_j \leq t < t_{j+1}$. For this purpose we use the DeBoor algorithm [31]:

$$\begin{aligned}\tilde{N} &= N + 1 \\ \tilde{k} &= k \\ \tilde{T} &= \{t_1, \dots, t_j, t, t_{j+1}, \dots, t_{N+k}\} \\ \alpha_i &= \begin{cases} 1 & 1 \leq i \leq j - k + 1 \\ \frac{t - t_i}{t_{i+k-1} - t_i} & j - k + 2 \leq i \leq j \\ 0 & j + 1 \leq i \end{cases} \\ \mathbf{Q}_i &= \alpha_i \mathbf{P}_i + (1 - \alpha_i) \mathbf{P}_{i-1}\end{aligned}$$

Many other algorithms exist, like blossoming for fast insertion algorithm. For more details about such topic, we refer to [86].

Order elevation

We can elevate the order of the basis, without changing the curve. Several algorithms exist for this purpose. We used the one by Huang et al. [64].

$$\begin{aligned}\tilde{k} &= k + m \\ \tilde{m}_i &= m_i + m \\ \tilde{N} &= N + ms\end{aligned}$$

Differential coefficients are defined as $\tilde{\mathbf{P}}_i^l$:

$$\tilde{\mathbf{P}}_i^l = \begin{cases} \tilde{\mathbf{P}}_i & l = 0 \\ \frac{1}{t_{i+p} - t_{i+l}} (\tilde{\mathbf{P}}_{i+1}^{l-1} - \tilde{\mathbf{P}}_i^{l-1}) & l > 0, t_{i+k-1} > t_{i+l} \\ 0 & l > 0, t_{i+k-1} = t_{i+l} \end{cases}$$

$$\beta_i = \sum_{l=1}^i m_l, 1 \leq i \leq s - 1, \text{ et } \alpha_i = \prod_{l=1}^i \frac{k-1-l}{k-1+m-l}, 1 \leq i \leq k - 2$$

We present the algorithm by [64]:

1. Compute $\tilde{\mathbf{P}}_0^j, 0 \leq j \leq k - 1$ et $\tilde{\mathbf{P}}_{\beta_l}^i, 1 \leq l \leq s - 1, k - m_l \leq i \leq k - 1$
2. Compute $\tilde{\mathbf{Q}}_0^j = \prod_{l=1}^j \binom{k-l}{k+m-l} \tilde{\mathbf{P}}_0^j, 0 \leq j \leq k - 1$
3. Compute $\tilde{\mathbf{Q}}_{\beta_l+m_l}^j = \prod_{l=1}^j \binom{k-l}{k+m-l} \tilde{\mathbf{P}}_{\beta_l}^j, 1 \leq l \leq s - 1, k - m_l \leq i \leq k - 1$
4. Compute $\tilde{\mathbf{Q}}_{\beta_l+m_l+i}^{k-1} = \tilde{\mathbf{Q}}_{\beta_l+m_l}^{(k-1)}, 1 \leq l \leq s - 1, 1 \leq i \leq m$
5. Compute $\tilde{\mathbf{Q}}_i^0$

Note that there exist other algorithms which expand the curve into a Bézier curve, then elevate the degree using Bernstein polynomials, finally come back to a description using *B-splines*. For more details, we refer to [93, 77]. The one given in [64] is more efficient and much more simple to implement. We can also use a more sophisticated version of this algorithm to insert new knots while elevating the degree.

1.3. Splines in CAD

1.3.3 NURBS

Let $\omega = (\omega_i)_{1 \leq i \leq N}$ be a sequence of non-negative reals. The NURBS functions are defined by a projective transformation:

Definition 1.3.4 (NURBS) The i -th NURBS of order k , associated to the knot vector T and the weights ω , is defined by

$$R_i^k = \frac{\omega_i N_i^k}{\sum_{j=1}^N \omega_j N_j^k}.$$

Notice that when the weights are equal to 1 the NURBS are B-splines.

Definition 1.3.5 (NURBS curve) The NURBS curve of order k associated to the knot vector T , the control points $(\mathbf{P}_i)_{1 \leq i \leq N}$ and the weights ω , is defined by

$$\mathbf{M}(t) = \sum_{i=1}^N R_i^k(t) \mathbf{P}_i$$

1.3.4 Multivariate tensor product NURBS

As for splines, one can define multivariate tensor product NURBS. For surfaces, we have the following definition.

Definition 1.3.6 (NURBS surface) The NURBS surface of order \mathbf{k} associated to the knot vectors $\{T^{(1)}, T^{(2)}\}$, the control points $(\mathbf{P}_{i,j})_{1 \leq i \leq N_1, 1 \leq j \leq N_2}$ and the weights $\{(\omega^{(1)}, \omega^{(2)})\}$, is defined by

$$\mathbf{M}(t^{(1)}, t^{(2)}) = \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} R_{i,j}(t^{(1)}, t^{(2)}) \mathbf{P}_{i,j}$$

with $R_{i,j}(t^{(1)}, t^{(2)}) = R_i^{(1)}(t^{(1)}) R_j^{(2)}(t^{(2)})$

Remark 1.3.7 NURBS functions inherit most of B-splines properties. Remark that in the interior of a knot span, all derivatives exist, and are rational functions with non vanishing denominator. We present here the definition of the perspective mapping. We construct the weighted control points $\mathbf{P}_i^\omega = (\omega_i x_i, \omega_i y_i, \omega_i z_i, \omega_i)$, then we define the B-spline curve in four-dimensional space as

$$\mathbf{M}^\omega(t) = \sum_{i=1}^N N_i^k(t) \mathbf{P}_i^\omega. \quad (1.3.25)$$

For fundamental geometric operations on NURBS curves, we use the latest transformation and algorithms on B-spline curves.

Remark 1.3.8 NURBS functions allow us to model, exactly, much more domains than B-splines. In fact, all conics can be exactly represented with NURBS. For more details, see [77].

1.3. Splines in CAD

	nature of the curve
$\omega_2 = 0$	line
$0 < \omega_2 < 1$	ellipse arc
$\omega_2 = 1$	parabolic arc
$\omega_2 > 1$	hyperbolic arc

Figure 1.7: Modeling conics using NURBS

1.3.5 Modeling conics using NURBS

In this section, we will show how to construct an arc of conic, using rational *B-splines*. Let us consider the following knot vector : $T = \{000\ 111\}$, the generated *B-splines* are Bernstein polynomials. The general form of a rational Bézier curve of degree 2 is:

$$\mathcal{C}(t) = \frac{\omega_1 N_1^2(t) \mathbf{P}_1 + \omega_2 N_2^2(t) \mathbf{P}_2 + \omega_3 N_3^2(t) \mathbf{P}_3}{\omega_1 N_1^2(t) + \omega_2 N_2^2(t) + \omega_3 N_3^2(t)} \quad (1.3.26)$$

Let us consider the case $\omega_1 = \omega_3 = 1$. Because of the multiplicity of the knots 0 and 1, the curve \mathcal{C} is linking the control point \mathbf{P}_1 to \mathbf{P}_3 . Depending on the value of ω_2 , we get different type of curves (figure 1.7) :

Circle

One can draw a circle using only 9 control points, and the parameters:

$N = 9, p = 2, T = \{000, \frac{1}{4}\frac{1}{4}, \frac{1}{2}\frac{1}{2}, \frac{3}{4}\frac{3}{4}, 111\}$. Control points and weights are given in the following table:

i	P_i	ω_i
1	(1, 0)	1
2	(1, 1)	$\frac{1}{\sqrt{2}}$
3	(0, 1)	1
4	(-1, 1)	$\frac{1}{\sqrt{2}}$
5	(-1, 0)	1
6	(-1, -1)	$\frac{1}{\sqrt{2}}$
7	(0, -1)	1
8	(1, -1)	$\frac{1}{\sqrt{2}}$
9	(1, 0)	1

Ring domain

To draw a circle inside another circle (*c.f* figure 1.8), we need the following parameters:

$$N_1 = 9, p_1 = 2, T^{(1)} = \{000, \frac{1}{4}\frac{1}{4}, \frac{1}{2}\frac{1}{2}, \frac{3}{4}\frac{3}{4}, 111\}$$

and

$$N_2 = 3, p_2 = 1, T^{(2)} = \{00, \frac{1}{2}, 11\}.$$

1.4. Splines in Approximation Theory

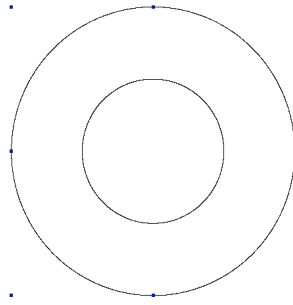


Figure 1.8: Domain plot and the exterior control points.

Ellipse

To draw an ellipse we can change the scaling of control points coordinates.

Modeling a Tokamak

Starting with an ellipse, one can modify the control points and the weights to have a simplified model of tokamak (*c.f.* figure 1.9). More examples can be found in the chapter 7, where we construct for MHD equilibrium, a mesh which is aligned to the magnetic field lines.

In practice, we will need to approximate domains. We can refer to the works of [83, 97].

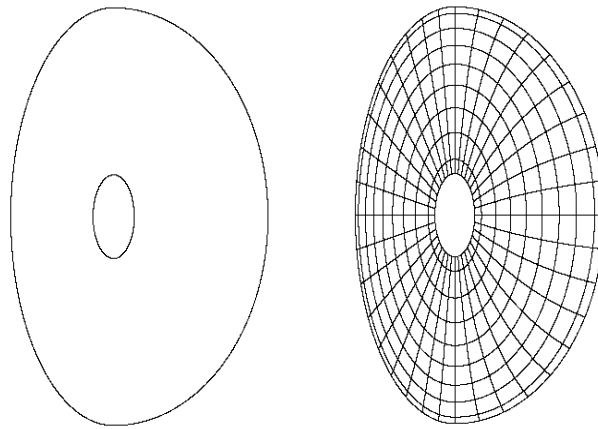


Figure 1.9: Poloidal plane designing of a tokamak. (left) a tokamak model, (right) tokamak after h -refinement.

1.4 Splines in Approximation Theory

Before giving the important results, it is very interesting to notice that we can approach the B -splines coefficients, using the value at some knots. Let us define $\bar{t}_j = \frac{t_{j+1} + \dots + t_{j+k-1}}{k-1}$, for each $j \in \{1, \dots, N+1\}$. Then, for each $S \in \mathcal{S}_k(T, I) \cap \mathcal{C}^1$ we have,

$$|S(\bar{t}_j) - [S]^j| \leq C(k)h^2 \|D^2 S\|_{[t_{j+1}, t_{j+k-1}]} \quad (1.4.27)$$

1.4. Splines in Approximation Theory

where $h := \max_{i \in \Lambda} (t_{i+1} - t_i)$.

If we define $VS = \sum_{i \in \Lambda} S(\bar{t}_i)N_i$, then we have

$$\|S - VS\| \lesssim h^2, \quad \forall S \in \mathcal{S}_k(T, I) \cap \mathcal{C}^m, \quad m \geq 2 \quad (1.4.28)$$

In the sequel, we explain how to recover the classical order $p + 1$, obtained using polynomial approximation. To do so, we shall use strategies from Functional Analysis.

Let $\{\mu_i\}_{1 \leq i \leq n}$ be linear forms on the space of continuous functions, such that:

$$|\mu_i g| \leq \|\mu_i\| \|g\|_{[t_i, t_{i+k}]}, \quad \forall g \in \mathcal{C}^0 \quad (1.4.29)$$

Let :

$$Ag = \sum_{i=1}^n (\mu_i g) N_i, \quad \forall g \in \mathcal{C}^0 \quad (1.4.30)$$

Therefore, if the operator A is preserving polynomials of degree less than $k - 1$, i.e $Au = u$, $\forall p \in \Pi_{<k}$, we have:

$$\|g - Ag\| \leq (1 + \max_i \|\mu_i\|) C(k) \|D^k g\| h^k, \quad \forall g \in \mathcal{C}^k \quad (1.4.31)$$

1.4.1 Quasi-interpolant

In this section, we define a quasi-interpolant, using the dual-functions, for a local approximation,

$$\pi_{\mathcal{S}_h} g = \sum_{i=1}^n (\lambda_i g) N_i \quad (1.4.32)$$

where dual functionals λ_i are given by 1.2.13.

We verify that it reproduces splines in \mathcal{S}_h :

$$\pi_{\mathcal{S}_h} s = s, \quad \forall s \in \mathcal{S}_h \quad (1.4.33)$$

Moreover, for each $g \in \mathcal{C}^{k-1}$, we have,

$$\|D^j g - D^j \pi_{\mathcal{S}_h} g\| \leq C(k, j) (m_t^{(2j-k)_+}) h^{k-j-1} \omega(D^{k-1} g; h)$$

with $m_t = \max\{\frac{\Delta t_r}{\Delta t_s}, |r - s| = 1, k \leq r, s \leq n\}$.

To extend the definition of the quasi-interpolant to functions in L^p , we must use the Hahn-Banach extension of the functionals λ_i (for more details we refer to [33]). Later in chapter 8, we shall use this idea to construct a quasi-interpolant for Box-splines.

1.4.2 Distance of a regular function to \mathcal{S}

In Approximation Theory, Jackson's inequality makes the relation between the distance of a regular function (function's best approximation) and the modulus of continuity of its derivatives.

Proposition 1.4.1 (Jackson inequality) :

Let $0 \leq j \leq k - 1$, $g \in \mathcal{C}^{(j)}$ then :

$$\mathbf{dist}(g, \mathcal{S}_T^k) \leq C(k, j) h^j \omega(D^j g; h)$$

In the case where $g \in \mathcal{C}^{(k)}$ we have :

$$\mathbf{dist}(g, \mathcal{S}_T^k) \leq C(k) h^k \|D^k g\| \quad (1.4.34)$$

1.5 Web-splines

Historically, the Web-spline method is the first Finite element method based on *B-splines*, that takes into account the boundary. This is done thanks to the weight function. In this section, we shall only consider uniform *B-splines*. Most of results in this section, are from [58, 60, 74].

Definition 1.5.1 (uniform B-splines) *The uniform B-Spline b^n of degree n , is defined by the recurrence formula :*

$$b^n(x) = \int_{x-1}^x b^{n-1}(t) dt$$

which is equivalent to,

$$\frac{db^n}{dx}(x) = b^{n-1}(x) - b^{n-1}(x-1)$$

under the assumption : $b^n(0) = 0$

Remark that in the chapter 8, we will use the same idea to construct the box-splines. Another definition of uniform *B-splines* can be done using the knot vector $T = \{0, 1, \dots, n\}$.

In the sequel, we shall note $b_{k,h}^n(x) = b^n(\frac{x}{h} - k)$, for any real $h > 0$ and an integer $k \in \mathbb{Z}$. In some cases, we will omit the indices h and k when they are defined in the context. Functions $(b_{k,h}^n)_{k,n}$ are the B-Splines over the grid $h\mathbb{Z}$.

Definition 1.5.2 (Cardinal spline) *A cardinal spline of degree $\leq n$ and a mesh size h , is any linear combination :*

$$\sum_{k \in \mathbb{Z}} c_k b_{k,h}^n$$

where $(c_k)_k \in \mathbb{R}$ are a real valued sequence.

We recall this important result, which is used for Multigrid methods [61], but can also be used in Multiscale methods:

Proposition 1.5.3 (Refinement)

$$b_{k,h}^n(x) = \frac{1}{2^n} \sum_{i=0}^{n+1} \binom{i}{n+1} b_{2k+i,h/2}^n(x)$$

Proposition 1.5.4 (Convolution)

$$b_{k,h}^{n+m+1} = b_{k,h}^n \star b_{k,h}^m$$

Proposition 1.5.5 (Scalar Product)

$$(b_{i,h}^n, b_{j,h}^n) = hb^{2n+1}(n+1+j-i) = s_h^n(j-i)$$

$$\left(\frac{d}{dx} b_{i,h}^n, \frac{d}{dx} b_{j,h}^n\right) = d_h^n(j-i) = \frac{1}{h}(2s^{n-1}(j-i) - s^{n-1}(j-i+1) - s^{n-1}(j-i-1)) = d^n(j-i)$$

the scalar product was computed over \mathbb{R} .

1.5. Web-splines

1.5.1 B-Splines on bounded domains

In the sequel, Ω will denote a real bounded domain in \mathbb{R}^n .

$$\mathbb{B}_h^n(\Omega) = \left\{ \sum_{k \in K} c_k b_{k,h}^n / (c_k)_{k \in K} \in \mathbb{R}, \forall k \in K : \text{supp}(b_{k,h}^n) \cap \Omega \neq \emptyset \right\}$$

For an easy reading, we shall write, when there is no ambiguity, $\mathbb{B} = \mathbb{B}_h^n(\Omega)$ et $b_k = b_{k,h}^n$ for fixed n and h .

Proposition 1.5.6 *Any multi-variate polynomial of degree n , can be represented in the domain Ω , using a cardinal spline of degree at most n :*

$$\sum_{k \in \mathbb{Z}} \alpha(k) b_{k,h}^n$$

where α is a multi-variate polynomial of degree at most n , for each variable k_μ

Proposition 1.5.7 (Local Linear Independance) *For any open $\Omega' \subset \Omega$, B-Splines b_k such that $\Omega' \cap \text{supp}(b_k) \neq \emptyset$ are linearly independant.*

1.5.2 Weight functions

Definition 1.5.8 *A weight function ω of order $p \in \mathbb{N}^*$ is a continuous function on $\bar{\Omega}$ such that $\omega(x) \simeq \text{dist}(x, \Gamma)^p$ for any $x \in \Omega$ and $\Gamma \subset \partial\Omega$. Where Γ admits a positive and sufficiently regular measure such that the distance function have a bounded gradient. If ω is regular and non vanishing on all the boundary, i.e $\partial\Omega$ and $p = 1$, and ω is said to be a standard weight function.*

Under these assumptions, we will notice $\omega \mathbb{B}_h^n(\Omega) = \text{span}_{k \in K} \omega b_k$.

Definition 1.5.9 *The weight function of order p , is said to be l -regular if :*

$$|\partial^k \omega(x)| \leq C(\omega) \text{dist}(x, \Gamma)^{p-|k|}, \forall |k| \leq \min(p, l)$$

Proposition 1.5.10 (R-functions) *For the basic operations on domains, we shall define the following weight functions operators:*

- *Complementary* : $r_C(\omega) = -\omega$
- *Intersection* : $r_{\Omega_1 \cap \Omega_2}(\omega_1, \omega_2) = \omega_1 + \omega_2 - \sqrt{\omega_1^2 + \omega_2^2}$
- *Union* : $r_{\Omega_1 \cup \Omega_2}(\omega_1, \omega_2) = \omega_1 + \omega_2 + \sqrt{\omega_1^2 + \omega_2^2}$

1.5.3 Web-Splines

Remarks that when a cardinal spline vanishes on a cell, it is inevitably vanishing on all the domain. Therefore, we can not use them in a finite element method on a general domain, with for example a Dirichlet boundary condition. A first solution to this problem, is the notion of interior and exterior B-spline.

Each cell in the grid, is defined by the m-uplet which consists of the left bound of each interval.

Definition 1.5.11 *A B-spline is said to be interior when it has at least one cell in the interior of the domain. Otherwise, it is said to be exterior.*

1.5. Web-splines

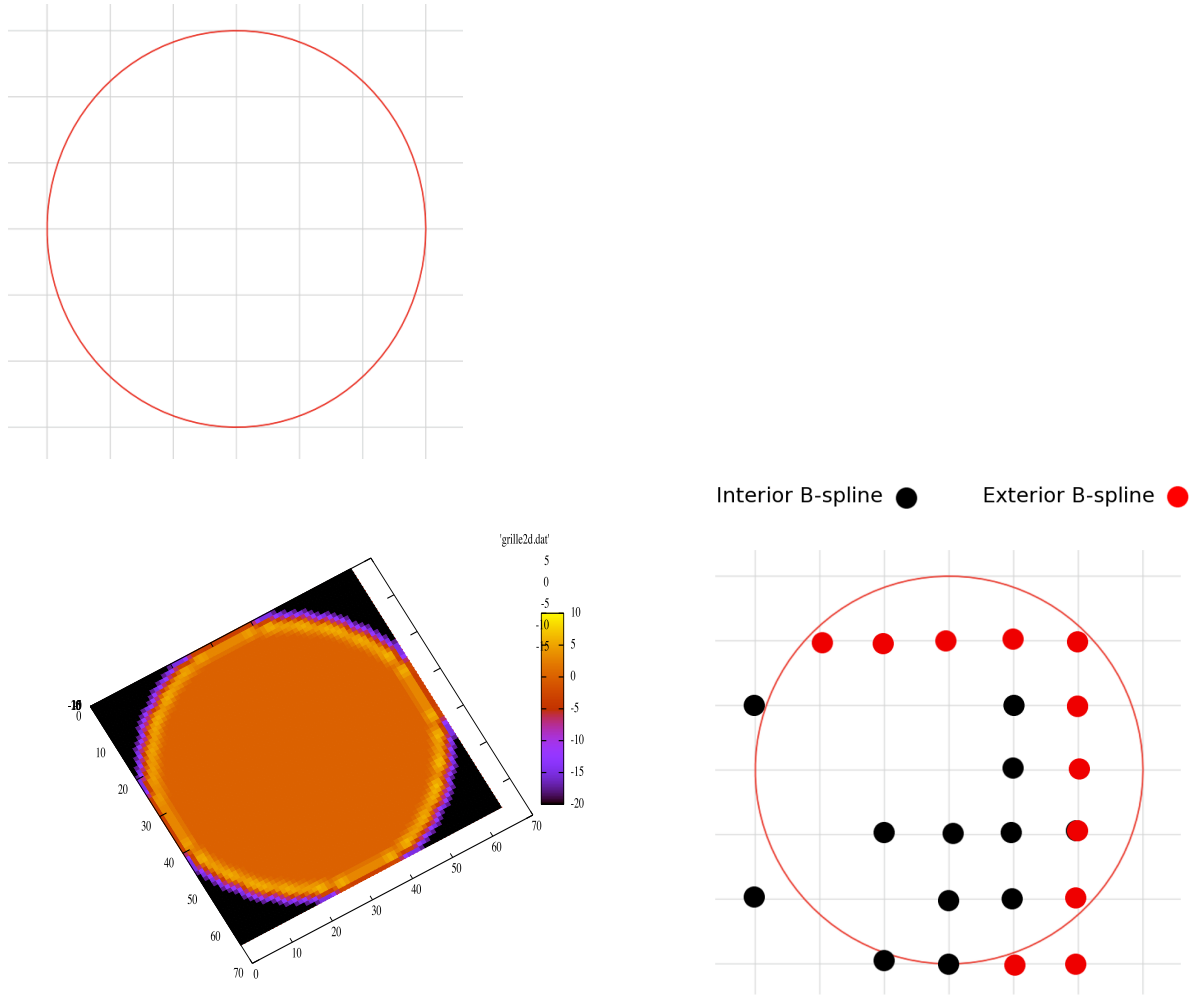


Figure 1.10: (1st line) : the computational domain is a circle centered at 0 and of radius 1. (2nd line) : (left) The number of links depending on the position of the spline, for a quadratic B-spline, on a 32×32 grid. (right) example of interior and exterior quadratic B-splines.

We denote \mathcal{I} the set of all interior B-splines. \mathcal{J} will denote the exterior ones. Figure 1.10 shows examples of interior and exterior quadratic B-splines in a circular domain.

As seen before, every polynomial P of degree n , defined on the domain Ω , can be written as :

$$\sum_{i \in \mathcal{I}} \alpha(i) b_i + \sum_{j \in \mathcal{J}} \alpha(j) b_j$$

where α is a multi-variate polynomial of degree at most n for each variable k_μ . This helps us to compute $\alpha(j), j \in \mathcal{J}$ from any cube $(n+1)^m$ of interior indices.

Let us denote, $\mathcal{I}(j) = i_j + \{0, \dots, n\}^m \subset \mathcal{I}$, where i_j is the nearest interior index to j . We determine α using Lagrange interpolation at the nodes $i \in \mathcal{I} \cap \mathcal{I}(j)$.

Hence, we get :

$$\alpha(j) = \sum_{i \in \mathcal{I}(j)} e_{i,j} \alpha(i)$$

1.5. Web-splines

for a given sequence $\{e_{i,j}\}$ to be determined. Therefore,

$$P(x) = \sum_{i \in \mathcal{I}} (b_i(x) + \sum_{j \in \mathcal{J}(i)} e_{i,j} b_j(x))$$

where, $\mathcal{J}(i) = \{j \in \mathcal{J} / i \in \mathcal{I}(j)\}$.

In this case, for a weight function ω , we denote:

$$B_i(x) = \frac{\omega(x)}{\omega(x_i)} (b_i(x) + \sum_{j \in \mathcal{J}(i)} e_{i,j} b_j(x))$$

where, x_i is the middle of the interior cell.

In what follows, we denote $w^e \mathbb{B}$ the set of all web-splines.

Remark 1.5.12 Usually, the majority of the coefficients $e_{i,j}$ are equal to 0. Notice that the choice of the interior index and the cube is very important. A wrong choice may lead to big coefficients $e_{i,j}$, and consequently will reduce the accuracy of the method. In this case, one may use a local refinement.

In figure 1.11, we plot the shape of basis functions depending on its position in the domain.

1.5.4 Stability in $w^e \mathbb{B}$

Proposition 1.5.13 For any weight function of order p , every linear combination of the interior web-splines verifies :

$$h^{\frac{m}{2}} \|C\| \lesssim \left\| \sum_{i \in \mathcal{I}} c_i B_i \right\|_0 \lesssim h^{\frac{m}{2}} \|C\|$$

We also have the following result:

Proposition 1.5.14 (Bernstein Inequality) If the weight function is of order p , and l -regular. For any $n \geq l$ we have:

$$h^k \left\| \sum_{i \in \mathcal{I}} c_i B_i \right\|_k \lesssim h^{\frac{m}{2}} \|C\|, \quad \forall k \leq l$$

1.5.5 Study of an elliptic partial differential equation

In this section, we study an elliptic partial differential equation with a Dirichlet boundary condition.

The problem we try to solve is of the form:

Find $u \in H_0^1(\Omega)$, such that:

$$\begin{cases} -\nabla \cdot (A \nabla u) = f, & \Omega \\ u = 0, & \partial\Omega \end{cases} \quad (1.5.35)$$

We will suppose that both the source term f and the boundary of the domain $\Omega \subset \mathbb{R}^m$ are regular, in order that a solution exists. A denotes here a matrix function :

$$A(x, y) = \begin{pmatrix} a_{11}(x, y) & a_{21}(x, y) \\ a_{21}(x, y) & a_{22}(x, y) \end{pmatrix}$$

1.5. Web-splines

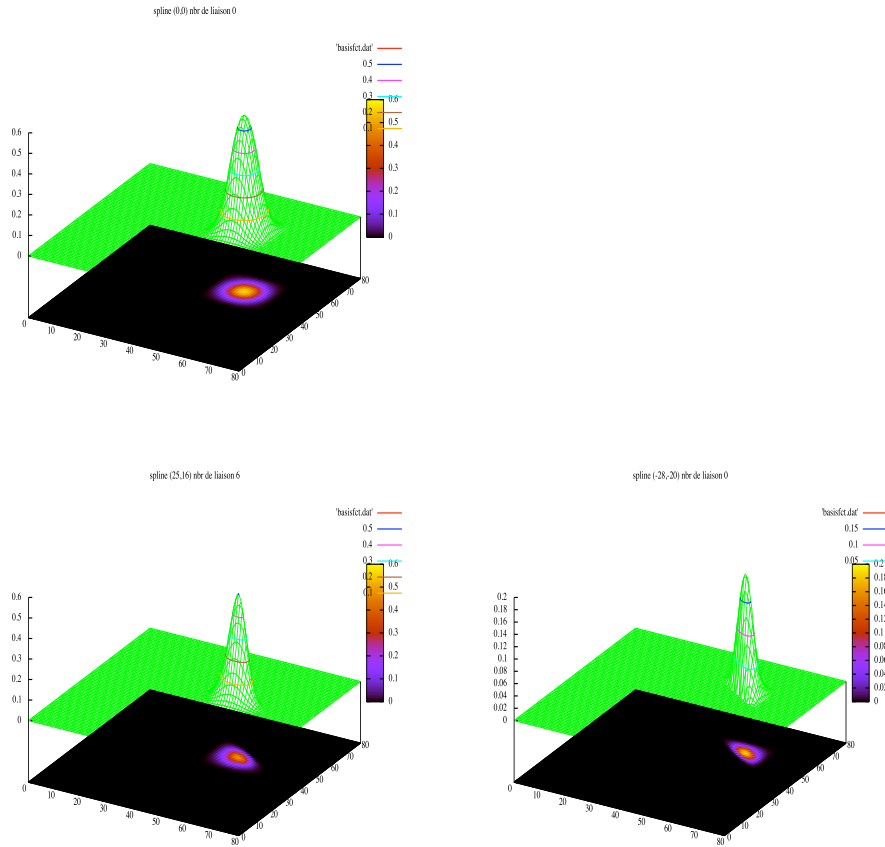


Figure 1.11: The web-spline support depending on the position of the index and the number of linked indices

The variational formulation leads to:

$$\int_{\Omega} A \nabla u \cdot \nabla \phi = \int f \phi, \quad \forall \phi \in H_0^1(\Omega)$$

As in the classical finite element method, we seek to approach the solution u by a function u_h on the finite dimensional space X_h .

$$\int_{\Omega} A \nabla u_h \cdot \nabla v_h = \int f v_h, \quad \forall v_h \in X_h$$

In our case, we will take for X_h , one of the following spaces \mathbb{B} , $w\mathbb{B}$ or $w^e\mathbb{B}$. Classically, by expanding both of u_h and v_h in the basis of $w^e\mathbb{B}$, we get:

$$\sum_{i,i'} [u_h]_i [v_h]_{i'} \int_{\Omega} A \nabla B_i \cdot \nabla B_{i'} = \sum_{i'} [v_h]_{i'} \int_{\Omega} B_{i'} f$$

By defining $M = (\int_{\Omega} A \nabla B_i \cdot \nabla B_{i'})_{i,i'}$ and $F = (\int_{\Omega} B_i f)_{i'}$, we get the classical linear system:

$$M [u_h] = F$$

1.5. Web-splines

1.5.6 Polynomial Approximation

We recall the classical result:

Proposition 1.5.15 (Bramble-Hilbert) *Let Π_n be the orthogonal projector into the space of polynomials of degree $\leq n$, on a domain hD . Therefore, for every $0 \leq q \leq p \leq n + 1$, we have*

$$|f - \Pi_n f|_{q,hD} \leq C(D, n) h^{p-q} |f|_{p,hD} \quad (1.5.36)$$

In the sequel, we will use the quasi-interpolant defined as:

$$P_h \cdot = \sum_{i \in \mathcal{I}} (B_i^*, \cdot)_0 B_i$$

where B_i^* are dual functions of B_i , see [58] for more details.

Proposition 1.5.16 *We have the following properties:*

- $\forall \pi \in \Pi_{<n}, \quad P_h(\omega\pi) = \omega\pi$
- *if ω is of order p and l -regular, then for each cell Q , and every $k \leq \min(l, n)$, we have:*

$$\|P_h u\|_{k, Q \cap \Omega} \lesssim h^{-k} \|u\|_{0, Q^*}$$

where $Q^* = \cup_{i \in \mathcal{I}(Q)} \text{supp}(B_i) \subset \Omega$ is the union of the supports of all web-splines that are non identically equal to 0 on $Q \cap \Omega$.

We have the following global convergence result:

Proposition 1.5.17 *If ω is of order p and l -regular, and if $v = \frac{u}{\omega}$ is regular on $\bar{\Omega}$, then $\forall k \leq \min(l, n)$, we have :*

$$\inf_{u_h \in \omega \mathbb{B}_h} \|u - u_h\|_k \leq \inf_{u_h \in \omega^e \mathbb{B}_h} \|u - u_h\|_k \lesssim h^{n+1-k}$$

1.5.7 Numerical results

We seek to solve the Poisson equation on a circular domain. The circle is centered at 0, with a radius equal to 1, which can be parametrized by the function:

$$\omega(x, y) = 1 - x^2 - y^2$$

Another approach given by Hollig, is to consider a weight function which is equal to 1 inside a particular subdomain, and then use the weight ω to take into account the boundary. We choose this subdomain to be a circle of radius equal to $1 - \delta$. Following the idea given by Hollig, and in order to have a regular weight function, we took :

$$\omega_0 := \omega_{\delta, \gamma}(x, y) = (1 - \mathbf{dist}(\mathbf{x}, \partial\Omega))^\gamma$$

where, $d(x, y) = \left(\frac{\sqrt{x^2 + y^2} - (1 - \delta)}{\delta}\right)^2$.

To compute element integrals, we used the Gauss-Lobatto quadrature method. But we still have a problem near the boundary. In fact we need a particular quadrature method, to take into account the behavior of the boundary. This may be difficult and not easy in the case of complex domains or a moving boundary. Therefore in practice, the web-spline method will be less accurate because of the error made near the boundary. Also, another problem is the choice of the parameter δ . Remark that we have implemented a version of an ϵ -algorithm for the evaluation of integrals. But, it is not sufficient.

How to choose δ

The choice of the parameter δ is very important, as we can see in figure 1.12. We have computed using quadratic B-splines, and different values of δ , the L^2 and H^1 error norm, in the case of grids 128×128 , 256×256 and 512×512 .

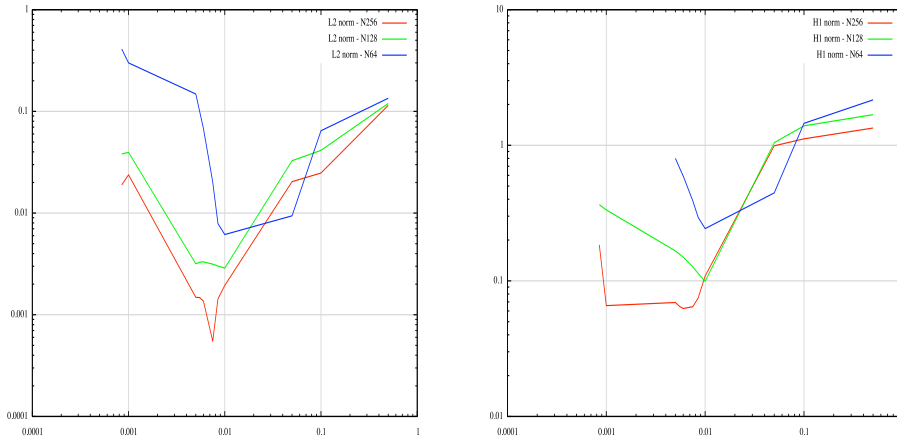


Figure 1.12: The impact of δ on the error norm. (left) L^2 norm, (right) H^1 norm

Another important thing is how do we link the exterior and interior indices. A bad algorithm may be bad on the accuracy of the method (see figure 1.13)

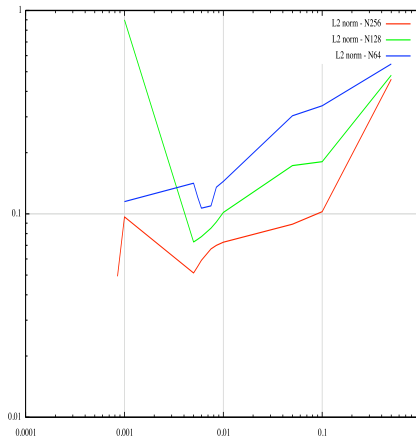


Figure 1.13: The impact of the linking algorithm on the L^2 error norm in the case of grids 128×128 , 256×256 and 512×512 .

To finish, another consequence of the precision-error of element integrals and the linking algorithm, is that the error $(u - u_h)$ will not preserve the symmetry of the domain. As shown in figure 1.14, we can see that on a circular domain, the error is very different comparing the north west to the south east.

1.6. Isogeometric analysis

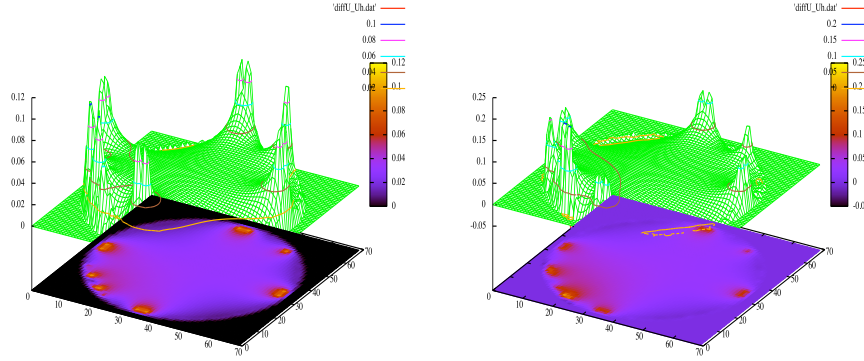


Figure 1.14: Examples of the error $u - u_h$ for quadratic B-splines

1.5.8 Conclusion

The web-spline method allows us to use uniform *B-Splines* in a Finite Element Method on general domains, by integrating an implicit definition of the boundary in the basis. However, the theoretical accuracy can not be achieved using simple quadrature formulas. As we do not want to have a specific formula for each domain, which will be unfeasible in Plasma Physics, the only solution to use *B-Splines*, would be to couple them with a mapping that maps the patch onto the real domain. The Web-spline method does not have only disadvantages! One of the most interesting points is the easy use of Multi-Grid Method thanks to the refinement strategy, even if we are penalized by the loss of accuracy.

1.6 Isogeometric analysis

The idea behind IGA method is to use the same functions that define the physical domain, to approach the solution of a partial differential equation. We will only treat the 2D case.

In the sequel, we consider 2 knot vectors $T_\xi = \{\xi_1, \dots, \xi_{N_1+p_1+1}\}$ and $T_\eta = \{\eta_1, \dots, \eta_{N_2+p_2+1}\}$. Let $W_\xi = \{\omega_1^\xi, \dots, \omega_{N_1}^\xi\}$ and $W_\eta = \{\omega_1^\eta, \dots, \omega_{N_2}^\eta\}$ be two weights sequences, and $(\mathbf{P}^{ij})_{1 \leq i \leq N_1, 1 \leq j \leq N_2}$ a sequence of control points. This defines a mapping

$$\mathbf{F}(\xi, \eta) = \sum_{1 \leq i \leq N_1, 1 \leq j \leq N_2} R_i^\xi(\xi) R_j^\eta(\eta) \mathbf{P}^{ij} \quad (1.6.37)$$

that maps the rectangular patch $[\xi_1, \xi_{N_1}] \times [\eta_1, \eta_{N_2}]$ onto the physical domain Ω . Where R^ξ and R^η are NURBS functions defined by knot vectors T_ξ and T_η , and weights W_ξ and W_η .

As said before, we consider only open knot vectors. Without loss of generality, we shall consider knot vectors of the form:

$$\xi_1 = \dots = \xi_{p_1+1} = \eta_1 = \dots = \eta_{p_2+1} = 0,$$

and

$$\xi_{N_1+1} = \dots = \xi_{N_1+p_1+1} = \eta_{N_2+1} = \dots = \eta_{N_2+p_2+1} = 1.$$

1.6. Isogeometric analysis

Let K be a cell in the physical domain. Q is the parametric associated cell and such that $K = \mathbf{F}(Q)$. Let $J_{\mathbf{F}}$ be the Jacobian of the transformation \mathbf{F} , that maps any parametric domain point (ξ, η) into physical domain point (x, y) (figure 1.15).

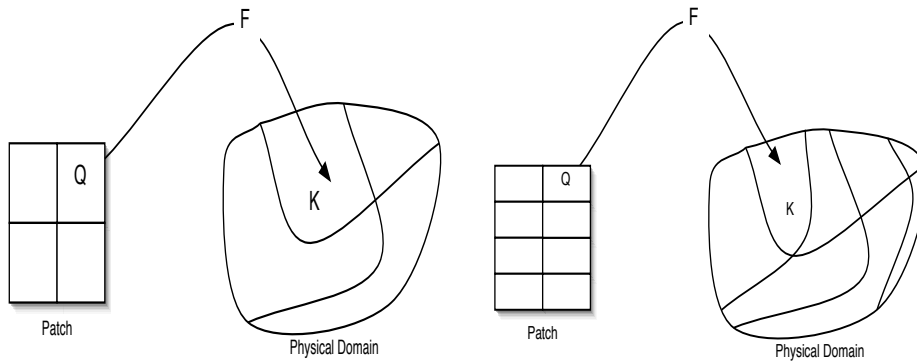


Figure 1.15: Mapping from the patch to the physical domain: (left) initial patch, (right) patch after h-refinement in the η direction. Here, we have $K = \mathbf{F}(Q)$

In figure 1.16, we give an example of the creation of a domain from a square.

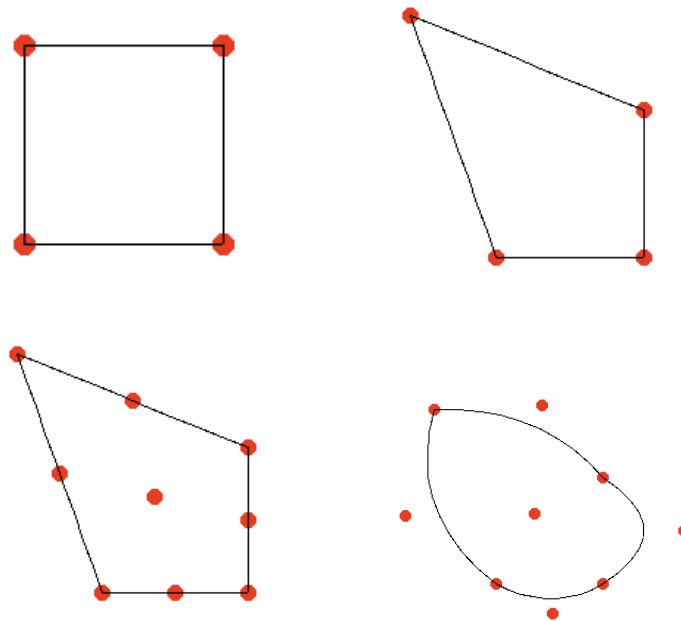


Figure 1.16: Creating and Meshing a domain starting from a square (from left to right and top to bottom) : we move a control point, then elevate the splines degree by 1, then moving the new control points.

For any function v of (x, y) we associate its representation in the parametric domain

$$\tilde{v}(\xi, \eta) := v \circ \mathbf{F}(\xi, \eta) = v(x, y).$$

The basis functions, *B-splines* (*NURBS*), will not be affected by these changes, as they

1.6. Isogeometric analysis

are invariant by affine transformations. For a point (x, y) in the physical domain, let us denote

$$(x, y) = \mathbf{F}(\xi, \eta), x = \alpha(\xi, \eta) \text{ and } y = \beta(\xi, \eta).$$

then,

$$\alpha_1 = \frac{\partial \alpha}{\partial \xi} \quad \alpha_2 = \frac{\partial \alpha}{\partial \eta} \quad \beta_1 = \frac{\partial \beta}{\partial \xi} \quad \beta_2 = \frac{\partial \beta}{\partial \eta},$$

we have for the determinant of the Jacobian

$$\det(J_{\mathbf{F}}) = \alpha_1 \beta_2 - \alpha_2 \beta_1, \quad (1.6.38)$$

$$J_{\mathbf{F}} = \begin{pmatrix} \alpha_1 & \alpha_2 \\ \beta_1 & \beta_2 \end{pmatrix}, \quad (1.6.39)$$

$$J_{\mathbf{F}}^{-1} = \frac{1}{\Delta} \begin{pmatrix} \beta_2 & -\alpha_2 \\ -\beta_1 & \alpha_1 \end{pmatrix}. \quad (1.6.40)$$

Let u be a (scalar or vector) function defined on the physical domain. When we use the patch coordinates, we will write it \tilde{u} , *idem* for the used spaces.

1.6.1 Refinement strategies

Refining the grid can be done in 3 different ways. This is the most interesting aspects of B-splines basis,

- using the patch parameter h , by inserting new knots. This is the h -refinement, it is the equivalent to mesh refinement of the classical finite element method.
- using the degree p , by elevating the B-spline degree. This is the p -refinement, it is the equivalent to using higher finite element order in the classical FEM.
- using the regularity of B-splines, by increasing / decreasing the multiplicity of inserted knots. This is the k -refinement. This new strategy does not have an equivalent in the classical FEM.

J.A. Evans et al. [39] studied the k -refinement using the theory of Kolmogorov n -widths. As we will see later, the use of this strategy can be more efficient than the classical p -refinement, as it reduces the dimension of the basis.

An active area of research is the study of local refinement. It is important to notice that the use of tensor products leads to the existence of a lot of superfluous control points, that might exist because of this cartesian grid in the parametric domain. Sederberg et al. [100] defined the notion of T-splines that allows us to reduce the number of those control points. In [35] Dörfler et al. used T-splines for local h -refinement in isogeometric analysis. Dokken, Kvamsdal and their team from SINTEF are currently developing another approach for local refinement, based on LR-splines [108]. In [114], authors propose the use of a hierarchical local refinement method.

1.6.2 Patch

The latest construction gives a coarse mesh, we can then, use $h/p/k$ refinements to create the grid. We can also use multiple patches to describe more complex domains [67, 23]. There are many ways to stick those patches together.

1.6. Isogeometric analysis

1.6.3 Grid generation

For this purpose, we use alternatively h and p -refinement. The minimal degree of basis functions is imposed by the domain design, which will be given by the coarsest mesh. When inserting knots, we can use uniformly-spaced knots or non uniformly-spaced ones. In figures 1.17 and 1.18, we give examples of such refinements.

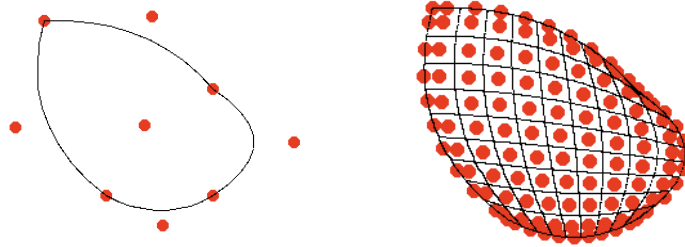


Figure 1.17: Grid generation: (left) the coarsest mesh, (right) domain after h -refinement. The minimal degree of basis functions, in this case, is 2.

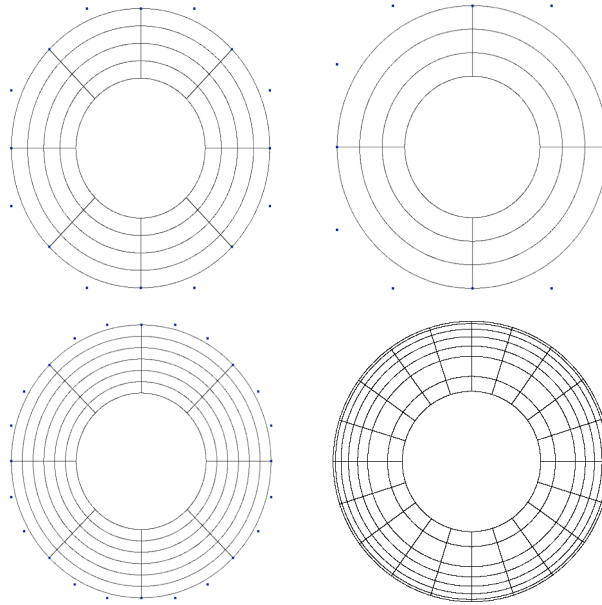


Figure 1.18: Grid generation: 1st line: (left) after h -refinement, (right) after p -refinement, $p_1 = p_2 = 3$. 2nd line: (left) after h -refinement $p_1 = p_2 = 3$, (right) using unstructured mesh, $p_1 = p_2 = 2$.

1.6.4 Local approximation

The first step, in finite element method is to study the quasi-interpolant $\pi_{\mathcal{S}_h}$. As noticed before, $\pi_{\mathcal{S}_h}$ preserves splines in \mathcal{S}_h . Another important property, is the stability of $\pi_{\mathcal{S}_h}$.

Theorem 1.6.1 (Quasi-interpolant stability) For each $v \in L^2(\mathcal{P})$ and $Q \in \mathcal{Q}_h$

$$\|\pi_{\mathcal{S}_h} v\|_{L^2(Q)} \lesssim \|v\|_{L^2(\bar{Q})} \quad (1.6.41)$$

1.6. Isogeometric analysis

As the quasi-interpolant verifies the preserving and stability properties, then we have the following local approximation result :

Theorem 1.6.2 (Bramble - Hilbert on S_h) *Let $0 \leq j \leq l \leq p + 1$ and $Q \in \mathcal{Q}_h$, then*

$$|v - \pi_{S_h} v|_{H^j(Q)} \lesssim h_Q^{l-j} |v|_{H^l(\bar{Q})} \quad \forall v \in \mathcal{H}_h^l(\bar{Q}) \cap L^2(\mathcal{P}) \quad (1.6.42)$$

Let us define the quasi-interpolant $\pi_{\mathcal{N}_h}$ on \mathcal{N}_h :

$$\pi_{\mathcal{N}_h} v := \frac{\pi_{S_h} \omega v}{\omega} \quad (1.6.43)$$

where $\omega = \sum_{1 \leq i \leq N_1, 1 \leq j \leq N_2} \omega_i^\xi(\xi) \omega_j^\eta(\eta) \mathbf{P}^{ij}$. The next result shows that $\pi_{\mathcal{N}_h}$ verifies a local approximation:

Theorem 1.6.3 (Bramble - Hilbert on \mathcal{N}_h) *Let $0 \leq j \leq l \leq p + 1$ and $Q \in \mathcal{Q}_h$, then*

$$|v - \pi_{\mathcal{N}_h} v|_{H^j(Q)} \lesssim h_Q^{l-j} |v|_{H^l(\bar{Q})} \quad \forall v \in \mathcal{H}_h^l(\bar{Q}) \cap L^2(\mathcal{P}) \quad (1.6.44)$$

In order to get the approximation in the physical domain, we need to estimate the change of variable using the mapping \mathbf{F} , [66]:

Lemma 1.6.4 *Let m be a non-negative integer, $Q \in \mathcal{Q}_h$ and $K = \mathbf{F}(Q)$. For all $v \in H^m(K)$, we have:*

$$|v \circ \mathbf{F}|_{H^m(Q)} \lesssim \|\det \nabla \mathbf{F}^{-1}\|_{L^\infty(K)}^{\frac{1}{2}} \sum_{j=0}^m \|\nabla \mathbf{F}\|_{L^\infty(Q)}^j |v|_{H^j(K)} \quad (1.6.45)$$

$$|v|_{H^m(K)} \lesssim \|\det \nabla \mathbf{F}\|_{L^\infty(Q)}^{\frac{1}{2}} \|\nabla \mathbf{F}\|_{L^\infty(Q)}^{-m} \sum_{j=0}^m |v \circ \mathbf{F}|_{H^j(Q)} \quad (1.6.46)$$

To extend the local approximation to the physical domain, we must define the quasi-interpolant on Ω :

$$\pi_{\mathcal{V}_h} v := \pi_{\mathcal{N}_h} (v \circ \mathbf{F}) \circ \mathbf{F}^{-1} \quad (1.6.47)$$

we give the local approximation result :

Theorem 1.6.5 (Bramble - Hilbert on \mathcal{V}_h) *Let $0 \leq j \leq l \leq p + 1$ and $K \in \mathcal{K}_h$, then*

$$|v - \pi_{\mathcal{V}_h} v|_{H^j(K)} \lesssim h_K^{l-j} \sum_{i=0}^l \|\nabla \mathbf{F}\|_{L^\infty(\bar{Q})}^{i-l} |v|_{H^i(\bar{K})} \quad \forall v \in \mathcal{H}^l(\bar{K}) \cap L^2(\Omega) \quad (1.6.48)$$

1.6.5 Global approximation using NURBS

With the previous properties, we can have a global approximation:

Theorem 1.6.6 (Global approximation) $\forall v \in H^l(\Omega), 0 \leq j \leq l \leq p + 1,$

$$\sum_{K \in \mathcal{K}_h} |v - \pi_{\mathcal{V}_h} v|_{\mathcal{H}_h^j(K)}^2 \lesssim \sum_{K \in \mathcal{K}_h} h_K^{2(l-j)} \sum_{i=0}^l \|\nabla \mathbf{F}\|_{L^\infty(\mathbf{F}^{-1}(K))}^{2(i-l)} |v|_{H^i(K)}. \quad (1.6.49)$$

more details can be found in [66].

Elliptic Equations

Contents

2.1	Galerkin-Ritz approximation	32
2.2	The variational formulation	34
2.3	Assembling matrices	35
2.3.1	Stiffness local matrix	35
2.3.2	Mass local matrix	36
2.3.3	Local load vector	36
2.3.4	Assembling matrices algorithm	36
2.4	Numerical results	37
2.4.1	Domain defined by B-splines curves	37
2.4.2	Solution for an affine transformation	37
2.5	Domain defined with NURBS curves	40
2.5.1	Poisson's equation on a quarter ring domain	40
2.5.2	Poisson's equation on a ring domain	42
2.6	Computing the solution on general domain	42
2.7	Nonlinear elliptic problems	45
2.7.1	Picard's algorithm	47
2.7.2	Newton's algorithm	47
2.7.3	Numerical results : Example from combustion theory	48

2.1. Galerkin-Ritz approximation

In this chapter, we will derive a variational formulation for elliptic partial differential equations, using the *IGA* paradigm. Details on the implementations are given, but we refer the reader to [65, 23, 48].

Let us consider the following problem:

For given functions f, g and k , find u such that:

$$\begin{cases} -\nabla \cdot (A\nabla u) = f & , \Omega \\ u = g & , \Gamma_D \\ \nabla u \cdot \mathbf{n} = k & , \Gamma_N \end{cases} \quad (2.0.1)$$

where $\Gamma_D \cup \Gamma_N = \partial\Omega$ and $\Gamma_D \cap \Gamma_N = \emptyset$.

Let us consider the following sets:

$$\mathcal{X} = H^1(\Omega)$$

$$\mathcal{S} = \{v/v \in H^1(\Omega), v|_{\Gamma_D} = g\}$$

$$\mathcal{V} = \{v/v \in H^1(\Omega), v|_{\Gamma_D} = 0\}$$

Using the Green formula, the variational formulation of (2.0.1) writes :

Find $u \in \mathcal{S}$ such that:

$$a(w, u) = l(w), \quad \forall w \in \mathcal{V} \quad (2.0.2)$$

where we denote :

$$a(w, v) = \int_{\Omega} A\nabla v \cdot \nabla w, \quad \text{and} \quad l(w) = \int_{\Omega} fw + \int_{\Gamma_N} kw \quad (2.0.3)$$

2.1 Galerkin-Ritz approximation

Let $\mathcal{X}_h, \mathcal{V}_h$ be finite dimensional subspaces of \mathcal{X}, \mathcal{V} , where h is a parameter intended to tend to zero. Let \mathcal{S}_h be a finite dimensional approximation of \mathcal{S} . Remark that the last set is not necessarily a vector space.

The discrete Galerkin formulation writes:

Find $u_h = v_h + g_h \in \mathcal{S}_h$ such that:

$$a(w_h, u_h) = l(w_h), \quad \forall w_h \in \mathcal{V}_h \quad (2.1.4)$$

2.1. Galerkin-Ritz approximation

where here g_h, k are given functions, with $g_h \in \mathcal{S}_h$.

We suppose a rearrangement of the basis functions such that: $R_i|_{\Gamma_D} = 0, \forall i \in \{1, \dots, n - n_D\}$. This helps us to construct a basis of \mathcal{V}_h i.e :

$$\forall w_h \in \mathcal{V}_h, w_h = \sum_{i=1}^{n-n_D} [w_h]^i R_i$$

$g^h \in \mathcal{S}_h$ is such that $[g_h]^i = 0, \forall i \in \{1, \dots, n - n_D\}$ and so, $g^h = \sum_{i=n-n_D+1}^n [g_h]^i R_i$
Therefore, $u_h \in \mathcal{S}_h$ writes

$$u_h = \sum_{i=1}^{n-n_D} [u_h]^i R_i + \sum_{i=n-n_D+1}^n [g_h]^i R_i .$$

Hence, equation (2.1.4) writes :

$$\begin{aligned} & \sum_{i=1}^{n-n_D} [w_h]^i \left\{ \sum_{j=1}^{n-n_D} [u_h]^j a(R_i, R_j) \right\} = \\ & \sum_{i=1}^{n-n_D} [w_h]^i \left\{ l(R_i) - \sum_{m=n-n_D+1}^n [u_h]^m a(R_i, R_m) \right\} \end{aligned}$$

therefore

$$\sum_{j=1}^{n-n_D} [u_h]^j a(R_i, R_j) = l(R_i) - \sum_{m=n-n_D+1}^n [g_h]^m a(R_i, R_m), \quad \forall i \in \{1, \dots, n - n_D\}$$

We denote for $i, j \in \{1, \dots, n - n_D\}$

$$\Sigma_{i,j} = a(R_i, R_j)$$

$$L_i = l(R_i) - a(R_i, g_h)$$

therefore we get:

$$\sum_{j=1}^{n-n_D} \Sigma_{i,j} [u_h]^j = L_i, \quad \forall i \in \{1, \dots, n - n_D\}$$

Finally, let us introduce the matrix :

$$\Sigma = (\Sigma_{i,j})_{1 \leq i, j \leq n-n_D}$$

and the vectors :

$$L = (L_i)_{1 \leq i \leq n-n_D}^T$$

$$[u_h] = ([u_h]^i)_{1 \leq i \leq n-n_D}^T$$

this leads to the linear system

$$\Sigma [u_h] = L .$$

2.2 The variational formulation

We have obtained the linear formulation of our discrete formulation. In the sequel, we are interested in rewriting it by computing the integrals on the parametric domain (patch). For each cell, Q in the physical domain, we can associate a cell \tilde{Q} in the parametric domain and such that $Q = \mathbf{F}(\tilde{Q})$. We denote by $J_{\mathbf{F}}$ the jacobian of this transformation \mathbf{F} , that maps each point (ξ, η) , in the parametric domain, onto a point (x, y) in the physical domain.

For each function v of (x, y) we associate the function

$$\tilde{v}(\xi, \eta) := v \circ \mathbf{F}(\xi, \eta) = v(x, y)$$

Notice that the basis functions R_i are not affected by this mapping. Using these notations, the reader can know at anytime, if we are working in the physical or the parametric domain.

Let us denote $J_{\mathbf{F}^{-1}}$ the jacobian of \mathbf{F}^{-1} . For each v, w defined on $\mathbf{F}(\tilde{Q}) = Q$, we have

$$\int_Q (A \nabla v) \cdot \nabla w \, dx \, dy = \int_{\tilde{Q}=\mathbf{F}^{-1}(Q)} (A J_{\mathbf{F}^{-1}} \tilde{\nabla} \tilde{v}) \cdot (J_{\mathbf{F}^{-1}} \tilde{\nabla} \tilde{w}) \frac{1}{\det J_{\mathbf{F}^{-1}}} \, d\xi \, d\eta$$

which can be written,

$$\int_Q (A \nabla v) \cdot \nabla w \, dx \, dy = \int_{\tilde{Q}} \tilde{\nabla} \tilde{v} J_{\mathbf{F}^{-1}}^T A^T J_{\mathbf{F}^{-1}} \tilde{\nabla} \tilde{w} \det(J_{\mathbf{F}}) \, d\xi \, d\eta$$

Let us denote $d = \det(J_{\mathbf{F}}) = \alpha_1 \beta_2 - \alpha_2 \beta_1$,

$$\begin{aligned} \alpha_1 &= \frac{\partial \alpha}{\partial \xi}, \alpha_2 = \frac{\partial \alpha}{\partial \eta} \\ \beta_1 &= \frac{\partial \beta}{\partial \xi}, \beta_2 = \frac{\partial \beta}{\partial \eta} \end{aligned}$$

where for simplification, we use for each point in the physical domain $(x, y) = \mathbf{F}(\xi, \eta)$,

$$x = \alpha(\xi, \eta) \quad \text{and} \quad y = \beta(\xi, \eta). \quad (2.2.5)$$

Doing this, $J_{\mathbf{F}^{-1}}$ writes simply,

$$J_{\mathbf{F}^{-1}} = \frac{1}{d} \begin{pmatrix} \beta_2 & -\beta_1 \\ -\alpha_2 & \alpha_1 \end{pmatrix}$$

In the sequel, let us denote $\Theta = J_{\mathbf{F}^{-1}}^T A^T J_{\mathbf{F}^{-1}}$, and let

$$\int_Q (A \nabla v) \cdot \nabla w \, dx \, dy = \int_{\tilde{Q}} \tilde{\nabla} \tilde{v} \, \Theta \, \tilde{\nabla} \tilde{w} \, \det(J_{\mathbf{F}}) \, d\xi \, d\eta$$

When A is the identity matrix, we get:

$$\Theta = \frac{1}{d^2} \begin{pmatrix} \alpha_2^2 + \beta_2^2 & -\alpha_1 \alpha_2 - \beta_1 \beta_2 \\ -\alpha_1 \alpha_2 - \beta_1 \beta_2 & \alpha_1^2 + \beta_1^2 \end{pmatrix}.$$

In the general case, if

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

2.3. Assembling matrices

we have:

$$\Theta = \frac{1}{d^2} \begin{pmatrix} \Theta_{11} & \Theta_{12} \\ \Theta_{21} & \Theta_{22} \end{pmatrix}$$

with,

$$\Theta_{11} = a_{11}\beta_2^2 - (a_{12} + a_{21})\alpha_2\beta_2 + a_{22}\alpha_2^2 \quad (2.2.6)$$

$$\Theta_{22} = a_{11}\beta_1^2 - (a_{12} + a_{21})\alpha_1\beta_1 + a_{22}\alpha_1^2 \quad (2.2.7)$$

$$\Theta_{21} = -(a_{11}\beta_1\beta_2 - a_{12}\alpha_1\beta_2 - a_{21}\alpha_2\beta_1 + a_{22}\alpha_1\alpha_2) \quad (2.2.8)$$

$$\Theta_{12} = -(a_{11}\beta_1\beta_2 - a_{12}\alpha_2\beta_1 - a_{21}\alpha_1\beta_2 + a_{22}\alpha_1\alpha_2) \quad (2.2.9)$$

For the load vector, the computation is much simpler,

$$\int_Q f v \, dx \, dy = \int_{\tilde{Q}} \tilde{f} \tilde{v} \, \det(J_{\mathbf{F}}) \, d\xi \, d\eta \quad (2.2.10)$$

And for the Neumann part,

$$\int_{\partial Q \cap \Gamma_N} k v \, d\Gamma = \int_{\partial \tilde{Q} \cap \tilde{\Gamma}_N} \tilde{k} \tilde{w} \, \det(J_{\mathbf{F}|_{\tilde{\Gamma}}}) \, d\tilde{\Gamma} \quad (2.2.11)$$

2.3 Assembling matrices

We have seen that

$$\sum_{j=1}^{n-n_D} \Sigma_{i,j} [u_h]^j = L_i, \quad \forall i \in \{1, \dots, n - n_D\}$$

with,

$$\begin{aligned} \Sigma_{i,j} &= a(R_i, R_j) = \int_{\tilde{\Omega}} \tilde{\nabla} R_i \Theta \tilde{\nabla} R_j \, \det(J_{\mathbf{F}}) \, d\xi \, d\eta \\ &= \sum_{e \in \mathcal{E}} \int_{\tilde{Q}_e} \tilde{\nabla} R_i \Theta \tilde{\nabla} R_j \, \det(J_{\mathbf{F}}) \, d\xi \, d\eta \\ &= \sum_{e \in \mathcal{E}} a_e(R_i, R_j) \end{aligned}$$

because $\{\tilde{Q}_e, e \in \mathcal{E}\}$ form a partition of the parametric domain \mathcal{P} , where \mathcal{E} is the set of all elements and

$$a_e(v, w) = \int_{\tilde{Q}_e} \tilde{\nabla} v \Theta \tilde{\nabla} w \, \det(J_{\mathbf{F}}) \, d\xi \, d\eta$$

2.3.1 Stiffness local matrix

For $(i_e, j_e) \in \{1, \dots, n_{en}\}$, we denote by

$$\Sigma_{i_e, j_e}^e = a_e(R_{i_e}, R_{j_e})$$

which defines a local stiffness matrix $\Sigma^e = (\Sigma_{i_e, j_e}^e)_{1 \leq i_e, j_e \leq n_{en}}$.

2.3. Assembling matrices

2.3.2 Mass local matrix

Assembling the mass matrix can be done following the same method. The mass matrix $M = (M_{i,j})_{1 \leq i,j \leq n-n_D}$, is defined as

$$M_{i,j} = \int_{\Omega} R_i R_j dx dy = \int_{\tilde{\Omega}} R_i R_j \det(J_{\mathbf{F}}) d\xi d\eta$$

In the same way, we define the local mass matrix, $M^e = (M_{i_e,j_e}^e)_{1 \leq i_e,j_e \leq n_{en}}$ with

$$M_{i_e,j_e}^e = \int_{\tilde{Q}_e} R_i R_j \det(J_{\mathbf{F}}) d\xi d\eta$$

2.3.3 Local load vector

We have

$$L_i = l(R_i) - a(R_i, g_h) = \sum_{e \in \mathcal{E}} l_e(R_i) - a_e(R_i, g_h)$$

Let us define, $L_{i_e}^e = l_e(R_{i_e}) - a_e(R_{i_e}, g_h)$
this is the load local vector $L^e = (L_{i_e}^e)_{1 \leq i_e \leq n_{en}}$

2.3.4 Assembling matrices algorithm

In this section, we give an algorithm of how one can assemble, with low cost, the matrices and vectors involved in the linear system. In the sequel, we consider that we have the following routines:

- `build_stiffness_Local` : it computes for each element e the quantity Σ^e ,
- `build_Mass_Local` : it computes for each element e the quantity M^e ,
- `build_source_Local` : it computes for each element e the quantity L^e ,
- `Assembly_from_elt` : it copies the local matrices into the global matrices,
- `assembly_Field` : it computes all functions needed, at a given point in the parametric domain.

```
subroutine Assembly_Matrix ( ... )
! loop over all elements of non zero measure
do elt = 1, nel
  if ( measure(e) == 0) then
    cycle
  end if
! loop over quadrature points
do il = 1, GL + 1
do jl = 1, GL + 1
  call assembly_Field ( ... )
  call build_stiffness_Local ( ... )
  call build_mass_Local ( ... )
  call build_source_Local ( ... )
```


2.4. Numerical results

```

    end do
    end do
    call Assembly_from_elt ( ... )
end do
end subroutine Assembly_Matrix

```

The algorithm can be understood as :

1. We loop over each element of non zero measure,
2. For each element e , we loop over the quadrature points in the element,
3. We compute all functions needed (R_{i_e} , its derivatives, \dots)
4. We compute all local matrices, and vectors (source terms)
5. We copy all local matrices into the global ones, thanks to the **LM** array.

Remark 2.3.1 *In PyIGA , we perform a loop on elements, at the initialization, in order to assign a new id, and take into account only non zero elements. The interest here, is that all fields (c.f chapter D) will have a common indexation of elements.*

Remark 2.3.2 *In [68], Hughes et al. propose new quadrature rules that take into account the regularity of splines across elements. For the moment, PyIGA does not handle these type of quadrature rules; we use the Gauss-Legendre quadrature.*

2.4 Numerical results

For validation tests, it is important to use mappings for which we can compute the inverse analytically. To this purpose, we use affine transformations.

In the sequel, we present different tests, beginning with domains defined by B-splines curves, and then NURBS. For more details on how we solve the linear system, we refer the reader to the *PyIGA's* documentation.

2.4.1 Domain defined by B-splines curves

We begin our tests with a domain defined by B-splines curves. We start with a square, where the mapping is simply the identity, and then a parallelepiped.

In figure 2.1, we can see the convergence order of the error $u - u_h$, using splines of degree p . The expected order is $p + 1$.

2.4.2 Solution for an affine transformation

We consider, here, the case where the mapping can be written in the form:

$$\begin{pmatrix} x \\ y \end{pmatrix} = A \begin{pmatrix} \xi \\ \eta \end{pmatrix} + B = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} \xi \\ \eta \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

We suppose $N_1 = N_2 = 2$, $p_1 = p_2 = 1$ et $T_1 = T_2 = \{ 0 \ 0 \ 1 \ 1 \}$. We get then,

$$M(\xi, \eta) = \begin{pmatrix} x \\ y \end{pmatrix} = \sum_{i=1}^2 \sum_{j=1}^2 N_i^{(1)}(\xi) N_j^{(2)}(\eta) \mathbf{P}_{ij}$$

2.4. Numerical results

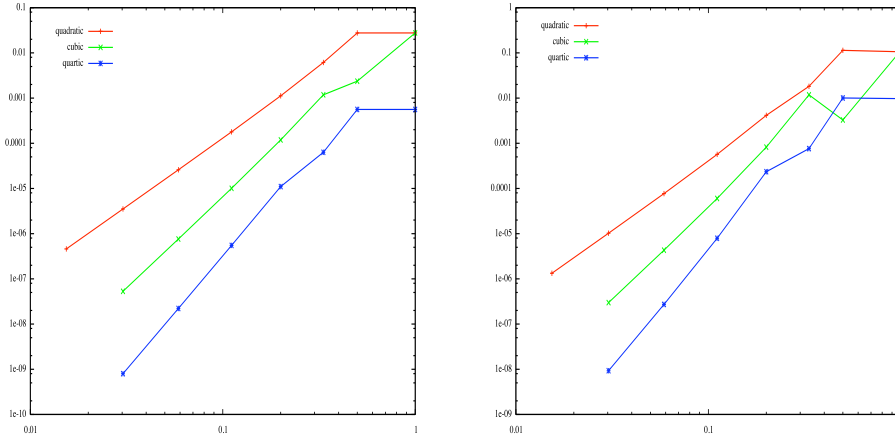


Figure 2.1: Convergence order: (left) for a square domain, (right) parallelepiped

Let us define $\mathbf{P}_{ij} = \begin{pmatrix} \mathbf{P}_{ij}^x \\ \mathbf{P}_{ij}^y \end{pmatrix}$. As we are using linear B-splines, we have:

$$\begin{cases} x = (1 - \xi)(1 - \eta)\mathbf{P}_{11}^x + \xi(1 - \eta)\mathbf{P}_{21}^x + (1 - \xi)\eta\mathbf{P}_{12}^x + \xi\eta\mathbf{P}_{22}^x \\ y = (1 - \xi)(1 - \eta)\mathbf{P}_{11}^y + \xi(1 - \eta)\mathbf{P}_{21}^y + (1 - \xi)\eta\mathbf{P}_{12}^y + \xi\eta\mathbf{P}_{22}^y \end{cases}$$

We want to have

$$\begin{cases} x = a_{11}\xi + a_{12}\eta + b_1 \\ y = a_{21}\xi + a_{22}\eta + b_2 \end{cases}$$

reordering the first equation leads to,

$$a_{11}\xi + a_{12}\eta + b_1 = \mathbf{P}_{11}^x + (\mathbf{P}_{21}^x - \mathbf{P}_{11}^x)\xi + (\mathbf{P}_{12}^x - \mathbf{P}_{11}^x)\eta + (\mathbf{P}_{11}^x - \mathbf{P}_{12}^x - \mathbf{P}_{21}^x + \mathbf{P}_{22}^x)\xi\eta$$

where,

$$\begin{cases} \mathbf{P}_{11}^x = b_1 \\ \mathbf{P}_{12}^x = a_{12} + b_1 \\ \mathbf{P}_{21}^x = a_{11} + b_1 \\ \mathbf{P}_{22}^x = a_{11} + a_{12} + b_1 \end{cases}$$

We also have :

$$\begin{cases} \mathbf{P}_{11}^y = b_2 \\ \mathbf{P}_{12}^y = a_{22} + b_2 \\ \mathbf{P}_{21}^y = a_{21} + b_2 \\ \mathbf{P}_{22}^y = a_{21} + a_{22} + b_2 \end{cases}$$

Posing $\delta = a_{11}a_{22} - a_{12}a_{21}$, we get

$$A^{-1} = \frac{1}{\delta} \begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix}$$

so that

$$\begin{cases} \xi = \frac{1}{\delta} \{a_{22}(x - b_1) - a_{12}(y - b_2)\} \\ \eta = \frac{1}{\delta} \{-a_{21}(x - b_1) + a_{11}(y - b_2)\} \end{cases}$$

We consider a particular solution of Poisson's equation, under Dirichlet's boundary condition :

$$u(x, y) = \tilde{u}(\xi, \eta) = \sin(\lambda_1\xi) \sin(\lambda_2\eta)$$

2.4. Numerical results

with $\lambda_1 = \frac{\pi}{L_1}$ et $\lambda_2 = \frac{\pi}{L_2}$, where $0 \leq \xi \leq L_1$ and $0 \leq \eta \leq L_2$. Here, $L_1 = L_2 = 1$ and $\lambda := \lambda_1 = \lambda_2$.

We denote

$$\begin{cases} \omega_1(x, y) = \lambda_1 \xi \\ \omega_2(x, y) = \lambda_2 \eta \end{cases}$$

where $u(x, y) = \sin(\omega_1(x, y)) \sin(\omega_2(x, y))$, then

$$\frac{\partial u}{\partial \xi} = \frac{\partial \omega_1}{\partial \xi} \cos(\omega_1) \sin(\omega_2) + \frac{\partial \omega_2}{\partial \xi} \cos(\omega_2) \sin(\omega_1)$$

and

$$\begin{aligned} \frac{\partial^2 u}{\partial \xi^2} &= \frac{\partial^2 \omega_1}{\partial \xi^2} \cos(\omega_1) \sin(\omega_2) + \frac{\partial^2 \omega_2}{\partial \xi^2} \cos(\omega_2) \sin(\omega_1) \\ &\quad - \left\{ \left(\frac{\partial \omega_1}{\partial \xi} \right)^2 + \left(\frac{\partial \omega_2}{\partial \xi} \right)^2 \right\} \sin(\omega_1) \sin(\omega_2) \\ &\quad + 2 \left\{ \frac{\partial \omega_1}{\partial \xi} \frac{\partial \omega_2}{\partial \xi} \right\} \cos(\omega_1) \cos(\omega_2) \end{aligned}$$

therefore,

$$\begin{aligned} \Delta u &= \Delta \omega_1 \cos(\omega_1) \sin(\omega_2) + \Delta \omega_2 \cos(\omega_2) \sin(\omega_1) \\ &\quad - \{ \|\nabla \omega_1\|^2 + \|\nabla \omega_2\|^2 \} \sin(\omega_1) \sin(\omega_2) \\ &\quad + 2 \{ \nabla \omega_1 \cdot \nabla \omega_2 \} \cos(\omega_1) \cos(\omega_2) \end{aligned}$$

As $\Delta \omega_1 = \Delta \omega_2 = 0$, and

$$\nabla \omega_1 = \frac{1}{\delta} \begin{pmatrix} a_{22} \\ -a_{12} \end{pmatrix}, \quad \nabla \omega_2 = \frac{1}{\delta} \begin{pmatrix} -a_{21} \\ a_{11} \end{pmatrix}$$

we get,

$$\begin{aligned} \Delta u &= \frac{-1}{\delta^2} \{ a_{11}^2 + a_{12}^2 + a_{21}^2 + a_{22}^2 \} \sin(\omega_1) \sin(\omega_2) \\ &\quad - \frac{2}{\delta^2} \{ a_{11} a_{12} + a_{21} a_{22} \} \cos(\omega_1) \cos(\omega_2) \end{aligned}$$

Therefore, we only need to take the source term as $f = -\Delta u$.

Numerical results

We have made several tests, depending on the transformation. For the first one, (*Test (1)*), we have considered the case

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

for the second one, (*Test (2)*), we used

$$A = \begin{pmatrix} 1 & 1 \\ 2 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

2.5. Domain defined with NURBS curves

and for the third (*Test (3)*), we used

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Figure 2.2 shows that we get the expected convergence order, which is $p + 1$, when using splines of degree p . We remark also the similar behavior of the convergence order in the three tests.

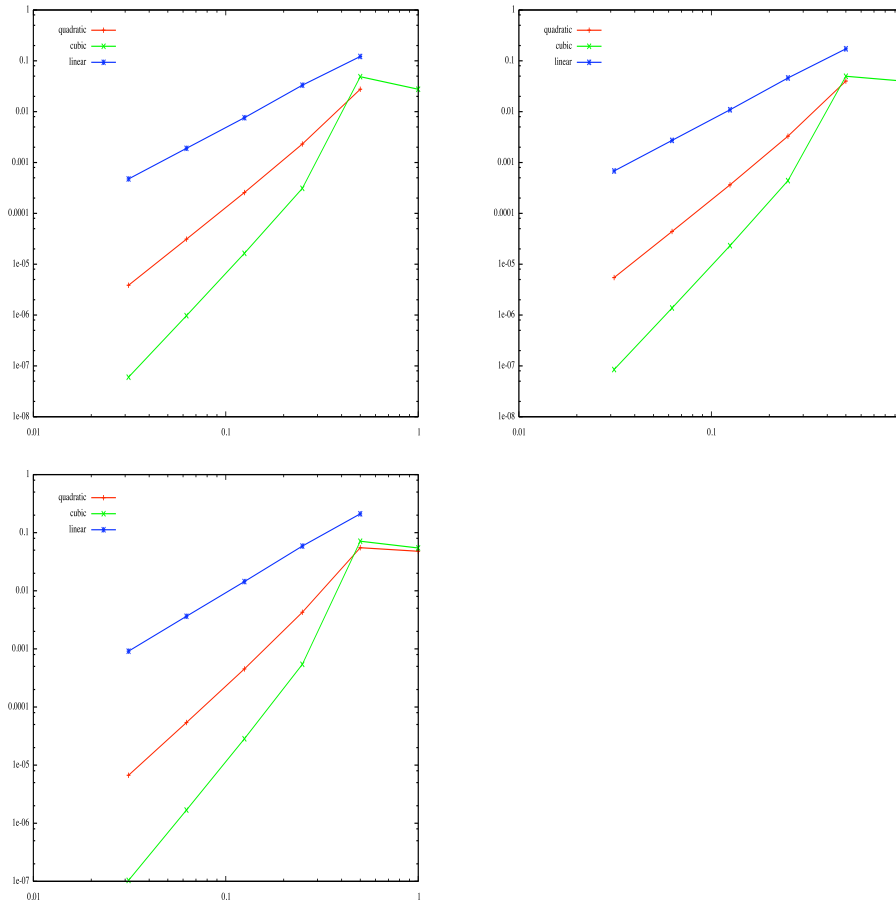


Figure 2.2: Convergence order for affine transformation. The L^2 norm error for, first line *Test (1)* (left) and *Test (2)* (right), second line *Test (3)*.

2.5 Domain defined with NURBS curves

2.5.1 Poisson's equation on a quarter ring domain

The first test is done for a quarter ring domain. The Dirichlet homogenous boundary condition was imposed on all the boundary. Notice that one can use different type of finite elements, thanks to the k -refinement strategy. Recalling that, when the multiplicity at a knot is m , for $1 \leq m \leq p$, the basis functions are then C^{p-m} at the interfaces of the involved elements.

2.5. Domain defined with NURBS curves

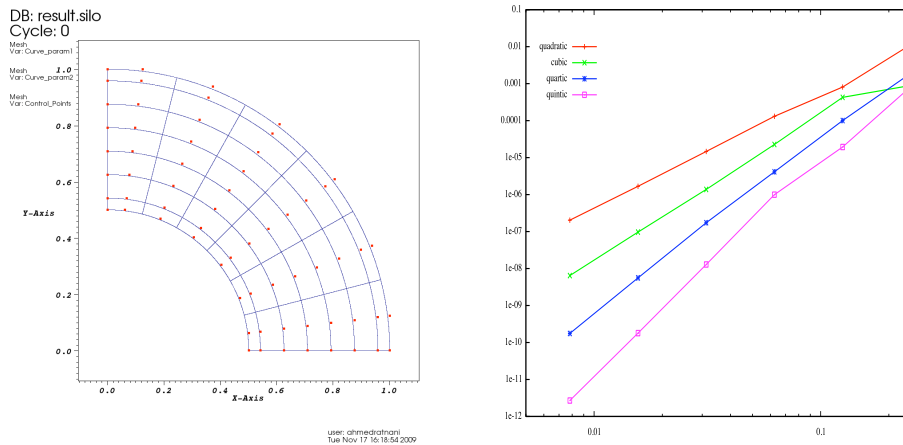


Figure 2.3: Quarter ring domain: (left) Domain and its control points, (right) Convergence order of the error $u - u_h$, in L^2 norm

Remark also, that increasing the multiplicity of inserted knots, increases the dimension of the discrete space \mathcal{V}_h . In figure 2.4, we plot the L^2 -error norm depending on the regularity of the finite space (multiplicity of inserted knots). Figure 2.5, shows the convergence

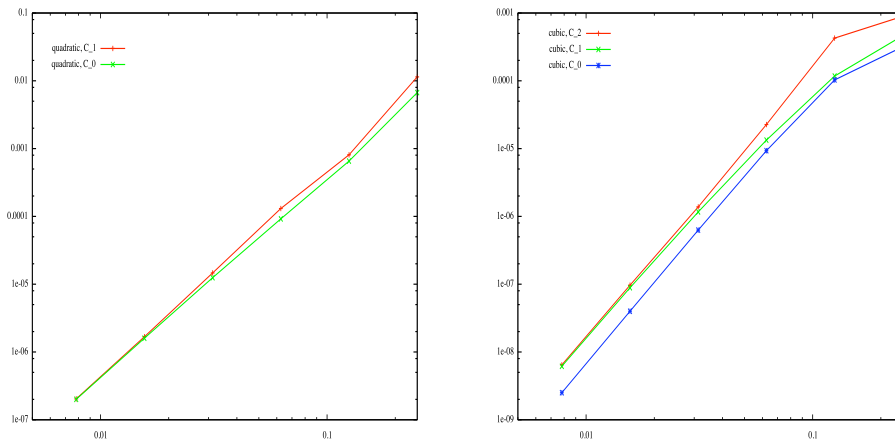


Figure 2.4: The error $u - u_h$, in the L^2 norm, depending on the regularity (using k-refinement)

order function of the number of degrees of freedom.

2.6. Computing the solution on general domain

degree	dof	L^2 -error	degree	dof	L^2 -error
2	4	$4.3922758 \cdot 10^{-2}$	3	9	$1.8776008 \cdot 10^{-2}$
2	16	$1.1411639 \cdot 10^{-2}$	3	25	$8.9466095 \cdot 10^{-4}$
2	64	$8.0964235 \cdot 10^{-4}$	3	81	$4.2640301 \cdot 10^{-4}$
2	256	$1.3062773 \cdot 10^{-4}$	3	289	$2.2630589 \cdot 10^{-5}$
2	1024	$1.4651313 \cdot 10^{-5}$	3	1089	$1.3793966 \cdot 10^{-6}$
2	4096	$1.6770547 \cdot 10^{-6}$	3	4225	$9.6672361 \cdot 10^{-8}$
2	16384	$2.0319858 \cdot 10^{-7}$	3	16641	$6.4468944 \cdot 10^{-9}$

Figure 2.5: The L^2 -error norm in function of the number of degrees of freedom, for quadratic and cubic splines

2.5.2 Poisson's equation on a ring domain

In this test, we look to solve the Poisson's problem in a ring domain. The analytical solution taken is $u(x, y) = \sin\left(\frac{\pi}{r_{max}^2 - r_{min}^2}(x^2 + y^2 - r_{min}^2)\right)$. Figures (2.6, 2.8) show the convergence order. In figure 2.7, we showed the L^2 -error norm as a function of the spline degree and the number of degrees of freedom.

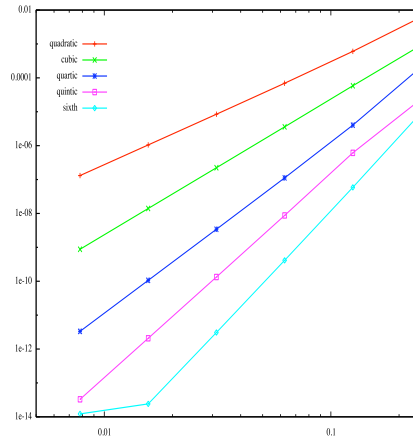


Figure 2.6: Convergence order of the error $u - u_h$ in the L^2 norm

2.6 Computing the solution on general domain

Let us denote: $d = \alpha_1\beta_2 - \alpha_2\beta_1$,

$$\alpha_1 = \frac{\partial\alpha}{\partial\xi}, \quad \alpha_2 = \frac{\partial\alpha}{\partial\eta}$$

$$\beta_1 = \frac{\partial\beta}{\partial\xi}, \quad \beta_2 = \frac{\partial\beta}{\partial\eta}$$

$$\alpha_{11} = \frac{\partial^2\alpha}{\partial\xi^2}, \quad \alpha_{12} = \frac{\partial^2\alpha}{\partial\xi\partial\eta}, \quad \alpha_{22} = \frac{\partial^2\alpha}{\partial\eta^2}$$

2.6. Computing the solution on general domain

degree	dof	L^2 -error	degree	dof	L^2 -error
2	16	$3.3114853 \cdot 10^{-2}$	3	36	$1.5201134 \cdot 10^{-2}$
2	48	$6.1729483 \cdot 10^{-3}$	3	80	$9.1622683 \cdot 10^{-4}$
2	128	$6.0494590 \cdot 10^{-4}$	3	180	$5.8446087 \cdot 10^{-5}$
2	384	$6.9280406 \cdot 10^{-5}$	3	476	$3.5830426 \cdot 10^{-6}$
2	1280	$8.4546119 \cdot 10^{-6}$	3	1452	$2.2346372 \cdot 10^{-7}$
2	4608	$1.0503232 \cdot 10^{-6}$	3	4940	$1.4003773 \cdot 10^{-8}$
2	17408	$1.3108596 \cdot 10^{-7}$	3	18060	$8.7738695 \cdot 10^{-10}$

Figure 2.7: The L^2 -error norm in function of the number of degrees of freedom, for quadratic and cubic splines

2	3	4	5	6
3.351082	3.970526	5.694030	5.265023	7.3045905
3.126287	4.027849	5.169361	6.133205	7.1552658
3.034637	4.003073	5.034454	6.035759	7.0688696
3.008905	3.996153	5.005171	6.002190	6.9955629
3.002248	3.996459	4.999353	5.981826	-

Figure 2.8: The convergence for different splines on grids 8×8 , 16×16 , 32×32 , 64×64 and 128×128

$$\beta_{11} = \frac{\partial^2 \beta}{\partial \xi^2}, \quad \beta_{12} = \frac{\partial^2 \beta}{\partial \xi \partial \eta}, \quad \beta_{22} = \frac{\partial^2 \beta}{\partial \eta^2}$$

In the sequel, we will give the solution for Poisson's problem, on a domain defined by a NURBS. We will look for a couple (u, f) such that u is the solution of the problem :

Find $u \in H_0^1(\Omega)$ such that:

$$-\Delta u = f \quad , \Omega \quad (2.6.12)$$

We have the following chain-rules formulas:

$$\frac{\partial \tilde{u}}{\partial \xi} = \alpha_1 \frac{\partial u}{\partial x} + \beta_1 \frac{\partial u}{\partial y}$$

$$\frac{\partial \tilde{u}}{\partial \eta} = \alpha_2 \frac{\partial u}{\partial x} + \beta_2 \frac{\partial u}{\partial y}$$

Also, for the second order, we have:

$$\frac{\partial^2 \tilde{u}}{\partial \xi^2} = \alpha_{11} \frac{\partial u}{\partial x} + \beta_{11} \frac{\partial u}{\partial y} + \alpha_1^2 \frac{\partial^2 u}{\partial x^2} + 2\alpha_1 \beta_1 \frac{\partial^2 u}{\partial x \partial y} + \beta_1^2 \frac{\partial^2 u}{\partial y^2}$$

$$\frac{\partial^2 \tilde{u}}{\partial \xi \partial \eta} = \alpha_{12} \frac{\partial u}{\partial x} + \beta_{12} \frac{\partial u}{\partial y} + \alpha_1 \alpha_2 \frac{\partial^2 u}{\partial x^2} + (\alpha_1 \beta_2 + \alpha_2 \beta_1) \frac{\partial^2 u}{\partial x \partial y} + \beta_1 \beta_2 \frac{\partial^2 u}{\partial y^2}$$

$$\frac{\partial^2 \tilde{u}}{\partial \eta^2} = \alpha_{22} \frac{\partial u}{\partial x} + \beta_{22} \frac{\partial u}{\partial y} + \alpha_2^2 \frac{\partial^2 u}{\partial x^2} + 2\alpha_2 \beta_2 \frac{\partial^2 u}{\partial x \partial y} + \beta_2^2 \frac{\partial^2 u}{\partial y^2}$$

2.6. Computing the solution on general domain

Let

$$H = \begin{pmatrix} \alpha_1^2 & 2\alpha_1\beta_1 & \beta_1^2 \\ \alpha_1\alpha_2 & \alpha_1\beta_2 + \alpha_2\beta_1 & \beta_1\beta_2 \\ \alpha_2^2 & 2\alpha_2\beta_2 & \beta_2^2 \end{pmatrix}$$

The inverse of H is easily computed:

$$H^{-1} = \frac{1}{d^2} \begin{pmatrix} \beta_2^2 & -2\beta_1\beta_2 & \alpha_2^2 \\ -\alpha_2\beta_2 & \alpha_1\beta_2 + \alpha_2\beta_1 & -\alpha_1\beta_1 \\ \beta_1^2 & -2\alpha_1\alpha_2 & \alpha_1^2 \end{pmatrix}$$

Let us denote:

$$D^2u = \begin{pmatrix} \frac{\partial^2 u}{\partial x^2} & \frac{\partial^2 u}{\partial x \partial y} & \frac{\partial^2 u}{\partial y^2} \end{pmatrix}^T \quad (2.6.13)$$

$$\tilde{D}^2\tilde{u} = \begin{pmatrix} \frac{\partial^2 \tilde{u}}{\partial \xi^2} & \frac{\partial^2 \tilde{u}}{\partial \xi \partial \eta} & \frac{\partial^2 \tilde{u}}{\partial \eta^2} \end{pmatrix}^T \quad (2.6.14)$$

and,

$$C = \begin{pmatrix} \alpha_{11} \frac{\partial u}{\partial x} + \beta_{11} \frac{\partial u}{\partial y} & \alpha_{12} \frac{\partial u}{\partial x} + \beta_{12} \frac{\partial u}{\partial y} & \alpha_{22} \frac{\partial u}{\partial x} + \beta_{22} \frac{\partial u}{\partial y} \end{pmatrix}^T \quad (2.6.15)$$

hence,

$$\tilde{D}^2\tilde{u} = HD^2u + C$$

therefore we get:

$$D^2u = H^{-1}(\tilde{D}^2\tilde{u} - C)$$

We can now compute the Laplacian of u , and we get :

$$\begin{aligned} d^2 \Delta u &= (\beta_1^2 + \beta_2^2) \left\{ \frac{\partial^2 \tilde{u}}{\partial \xi^2} - \alpha_{11} \frac{\partial u}{\partial x} - \beta_{11} \frac{\partial u}{\partial y} \right\} \\ &\quad - 2(\alpha_1\alpha_2 + \beta_1\beta_2) \left\{ \frac{\partial^2 \tilde{u}}{\partial \xi \partial \eta} - \alpha_{12} \frac{\partial u}{\partial x} - \beta_{12} \frac{\partial u}{\partial y} \right\} \\ &\quad + (\alpha_1^2 + \alpha_2^2) \left\{ \frac{\partial^2 \tilde{u}}{\partial \eta^2} - \alpha_{22} \frac{\partial u}{\partial x} - \beta_{22} \frac{\partial u}{\partial y} \right\} \end{aligned}$$

On the other hand, we have

$$\frac{\partial u}{\partial x} = \frac{1}{d} \left\{ \beta_2 \frac{\partial \tilde{u}}{\partial \xi} - \beta_1 \frac{\partial \tilde{u}}{\partial \eta} \right\}$$

$$\frac{\partial u}{\partial y} = \frac{1}{d} \left\{ -\alpha_2 \frac{\partial \tilde{u}}{\partial \xi} + \alpha_1 \frac{\partial \tilde{u}}{\partial \eta} \right\}$$

we inject these relations in the precedent equation, and we get:

$$\Delta u = c_{11} \frac{\partial^2 \tilde{u}}{\partial \xi^2} + 2c_{12} \frac{\partial^2 \tilde{u}}{\partial \xi \partial \eta} + c_{22} \frac{\partial^2 \tilde{u}}{\partial \eta^2} + c_1 \frac{\partial \tilde{u}}{\partial \xi} + c_2 \frac{\partial \tilde{u}}{\partial \eta} \quad (2.6.16)$$

2.7. Nonlinear elliptic problems

where :

$$c_{11} = \frac{1}{d^2} \{\beta_1^2 + \beta_2^2\} \quad (2.6.17)$$

$$c_{12} = -\frac{1}{d^2} \{\alpha_1 \alpha_2 + \beta_1 \beta_2\} \quad (2.6.18)$$

$$c_{22} = \frac{1}{d^2} \{\alpha_1^2 + \alpha_2^2\} \quad (2.6.19)$$

$$c_1 = \frac{-\alpha_{11}\beta_2 + \beta_{11}\alpha_2}{d} c_{11} + 2 \frac{-\alpha_{12}\beta_2 + \beta_{12}\alpha_2}{d} c_{12} + \frac{-\alpha_{22}\beta_2 + \beta_{22}\alpha_2}{d} c_{22} \quad (2.6.20)$$

$$c_2 = \frac{\alpha_{11}\beta_1 - \beta_{11}\alpha_1}{d} c_{11} + 2 \frac{\alpha_{12}\beta_1 - \beta_{12}\alpha_1}{d} c_{12} + \frac{\alpha_{22}\beta_1 - \beta_{22}\alpha_1}{d} c_{22} \quad (2.6.21)$$

Remark that those expressions can be computed easily, by deriving the position vector. We chose the solution u in the form : $u = \sin(\omega)$. As $\tilde{u} = \sin(\tilde{\omega})$, we get using (2.6.16):

$$\begin{aligned} \Delta u = \cos(\tilde{\omega}) \{ & c_{11} \frac{\partial^2 \tilde{\omega}}{\partial \xi^2} + 2c_{12} \frac{\partial^2 \tilde{\omega}}{\partial \xi \partial \eta} + c_{22} \frac{\partial^2 \tilde{\omega}}{\partial \eta^2} + c_1 \frac{\partial \tilde{\omega}}{\partial \xi} + c_2 \frac{\partial \tilde{\omega}}{\partial \eta} \} \\ & - \sin(\tilde{\omega}) \{ c_{11} (\frac{\partial \tilde{\omega}}{\partial \xi})^2 + 2c_{12} \frac{\partial \tilde{\omega}}{\partial \xi} \frac{\partial \tilde{\omega}}{\partial \eta} + c_{22} (\frac{\partial \tilde{\omega}}{\partial \eta})^2 \} \end{aligned}$$

Then, we need to take $f = -\Delta u$.

In figure 2.9, we plot the difference between the numerical and analytical solution. In figure 2.10, we plot particular solutions, in a turbulent mode.

2.7 Nonlinear elliptic problems

In this section, we present how one can solve nonlinear elliptic partial differential equations. Such problems arise in *MHD*-equilibrium [29, 9].

In the sequel, we shall consider the following problem:

Find u such that:

$$\begin{cases} -\nabla \cdot (A \nabla u) + Bu = F(\mathbf{x}, u) & , \Omega \\ u = 0 & , \partial \Omega \end{cases} \quad (2.7.22)$$

Typical problems in *MHD* equilibrium are of the form,

$$F(r, z, u) := -(r^2 f_1(u) + f_2(u)), \quad \text{for } \mathbf{x} = (r, z) \quad (2.7.23)$$

Let \mathcal{V}_h be the discrete space, such that $\mathcal{V}_h = \text{span}\{\varphi_b, b \in \Lambda\}$, then the variational formulation of (2.7.22) is :

$$\int_{\Omega} (A \nabla u) \cdot \nabla \varphi_b + \int_{\Omega} Bu \varphi_b = \int_{\Omega} F(\mathbf{x}, u) \varphi_b, \quad \forall b \in \Lambda \quad (2.7.24)$$

thus, by expanding u_h over \mathcal{V}_h , using $u_h = \sum_{b \in \Lambda} [u]^b \varphi_b$, we get :

$$\sum_{b \in \Lambda} [u]^b \left\{ \int_{\Omega} (A \nabla \varphi_b) \cdot \nabla \varphi_{b'} + \int_{\Omega} B \varphi_b \varphi_{b'} \right\} = \int_{\Omega} F(\mathbf{x}, u_h) \varphi_{b'}, \quad \forall b' \in \Lambda \quad (2.7.25)$$

this leads to solving the problem :

$$\mathcal{S}[u] = \mathcal{F}([u]) \quad (2.7.26)$$

2.7. Nonlinear elliptic problems

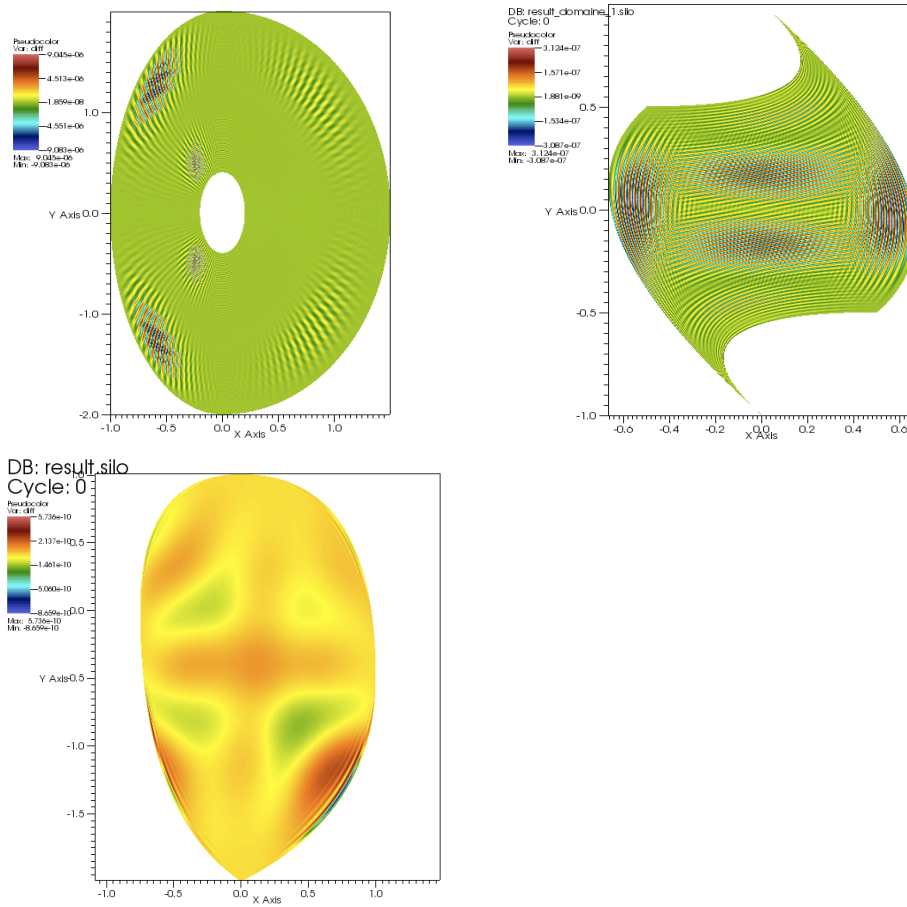
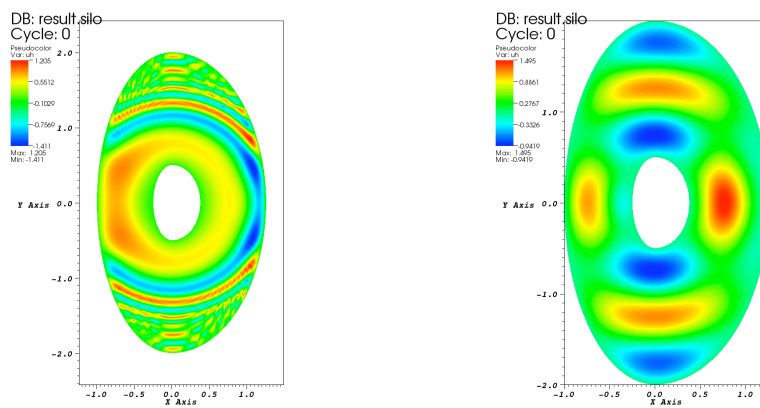


Figure 2.9: The difference between the numerical and analytical solution



user: ahmedrahani
Mon Sep 28 15:21:51 2009

user: ahmedrahani
Mon Sep 28 17:50:57 2009

Figure 2.10: Particular solutions, in a turbulent mode.

where,

$$\mathcal{S}_{b,b'} = \int_{\Omega} (A \nabla \varphi_b) \cdot \nabla \varphi_{b'} + \int_{\Omega} B \varphi_b \varphi_{b'}, \quad \forall b, b' \in \Lambda \quad (2.7.27)$$

$$\mathcal{F}([u])_{b'} = \int_{\Omega} F(\mathbf{x}, u_h) \varphi_{b'}, \quad \forall b' \in \Lambda \quad (2.7.28)$$

2.7. Nonlinear elliptic problems

2.7.1 Picard's algorithm

This is the simplest algorithm, and the less accurate also. To solve (2.7.26), we use the following algorithm

1. X^0 is given,
2. knowing X^n , we solve :

$$\mathcal{S}X^{n+1} = \mathcal{F}(X^n) \quad (2.7.29)$$

2.7.2 Newton's algorithm

Let us define the function :

$$g(X) = \mathcal{S}X - \mathcal{F}(X) \quad (2.7.30)$$

thus $[u]$ is a zero of the function g . To solve 2.7.26, we use Newton's method. As $J_{g(X)} = \mathcal{S} - J_{\mathcal{F}(X)}$, the Newton's method is:

- X^0 is given,
- knowing X^n , we solve :

$$J_{g(X^n)}(X^{n+1} - X^n) = -g(X^n) \quad (2.7.31)$$

The algorithm is the following:

1. we compute the mass matrix associated to the function : $\partial_u \mathcal{F}$, i.e :

$$M_{b,b'}^n = \int_{\Omega} \partial_u F(\mathbf{x}, \sum_{b \in \Lambda} X_b^n \varphi_b) \varphi_b \varphi_{b'} \quad (2.7.32)$$

2. compute the term $\mathcal{F}(X^n)$:

$$[\mathcal{F}(X^n)]_{b'} = \int_{\Omega} F(\mathbf{x}, \sum_{b \in \Lambda} X_b^n \varphi_b) \varphi_{b'} \quad (2.7.33)$$

3. compute $g(X^n)$:

$$g(X^n) = \mathcal{S}X^n - \mathcal{F}(X^n) \quad (2.7.34)$$

4. compute $J_{g(X^n)}$:

$$J_{g(X^n)} = \mathcal{S} - J_{\mathcal{F}(X^n)} = \mathcal{S} - M^n \quad (2.7.35)$$

5. solve $J_{g(X^n)}(X^{n+1} - X^n) = -g(X^n)$, and then find X^{n+1}

2.7. Nonlinear elliptic problems

2.7.3 Numerical results : Example from combustion theory

In this section, we shall solve the equation :

$$-\Delta u = -a \exp^{\beta u} \quad (2.7.36)$$

This example occurs in combustion theory, but also models the electrostatic potential in a charged body.

The general form of solutions is :

$$u(x, y) = \frac{1}{\beta} \ln \frac{8C}{a\beta} - \frac{2}{\beta} \ln |(x + A)^2 + (y + B)^2 - C| \quad (2.7.37)$$

for more solutions, we refer to [28].

In order to have the function u , vanishing at the boundary, we shall take the following values of parameters:

$$C = -\frac{1}{2}, \quad A = B = 0, \quad a\beta = -4$$

which gives,

$$u(x, y) = -\frac{2}{\beta} \ln |x^2 + y^2 + \frac{1}{2}| \quad (2.7.38)$$

One can easily check that u verifies:

$$-\Delta u = \frac{4}{\beta} \exp^{\beta u} \quad (2.7.39)$$

In the following test, we took $\beta = -1$.

To have homogeneous Dirichlet boundary condition, the domain will be a circle of radius $\frac{\sqrt{2}}{2}$, centered at 0.

In figure 2.12, we can see that, as predicted by the theory, Picard's algorithm is less accurate. We notice also, that in the case of Newton's algorithm, there exist oscillations for higher order. We also notice that up to the 10th iteration, the convergence is exponential. Notice that oscillations are less important for the 64 × 64 than 32 × 32 grid.

As in the practice we do not know the exact solution, we shall stop the algorithm when the error $\|X^{n+1} - X^n\|_{\infty}$ is less than a tolerance value. Using Newton's method, we achieve a precision of 0.00484 after only 8 iterations, using splines of degree 2 in 32 × 32 grid. For cubic splines, we notice that the algorithm is less stable, probably due to precision error, as the order of the Taylor expansion used in the Newton's algorithm is less than the spline's one.

2.7. Nonlinear elliptic problems

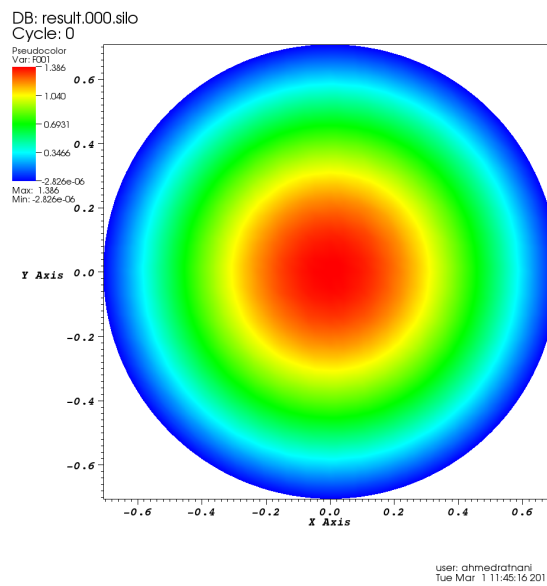


Figure 2.11: Plot of the solution u of (2.7.39)

Plot of the evolution of the L^2 -norm error using Newton-Raphson's algorithm

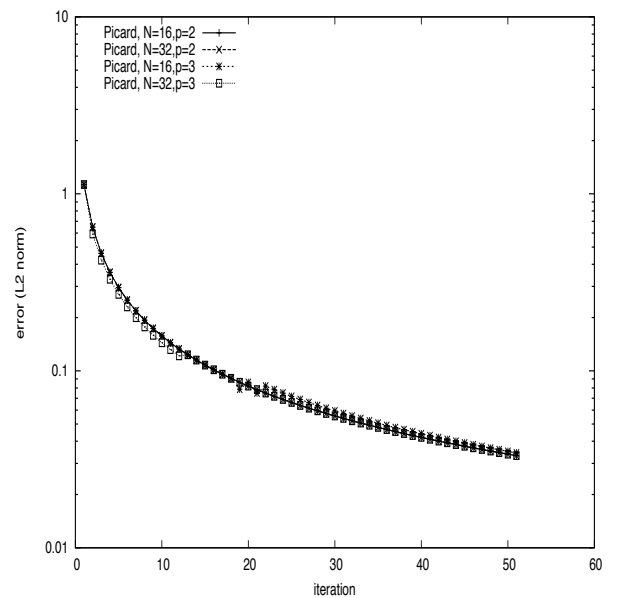
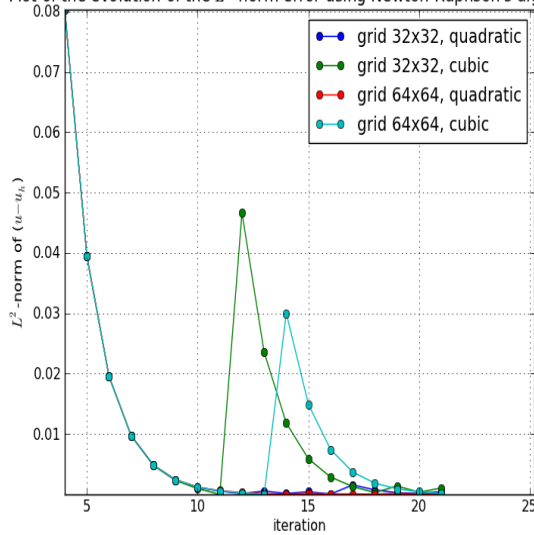


Figure 2.12: L^2 norm error, in function of the number of iterations. (left), using Newton's algorithm. (right), using Picard's algorithm

2.7. Nonlinear elliptic problems

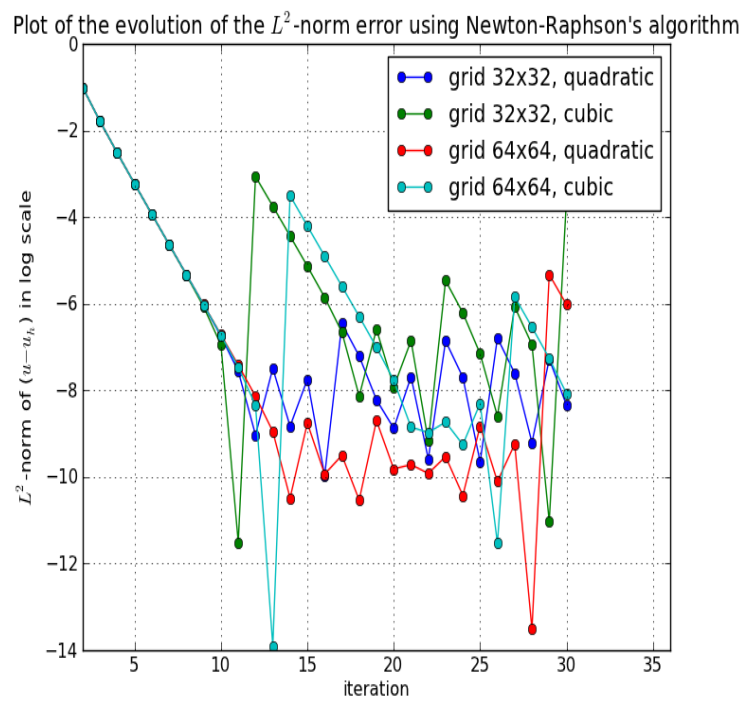


Figure 2.13: L^2 norm error, in log-scale, in function of the number of iterations using 9 Gauss-Legendre points per 1D-mesh.

Application to the Quasi-Neutral equation

Contents

3.1	Introduction	52
3.2	Quasi-neutrality equation	53
3.3	A fast solver for polar coordinates	54
3.4	Numerical validation	56
3.4.1	Test case 1: Order of convergence for Poisson in polar coordinates	56
3.4.2	Test case 2: Chaotic solution	58
3.5	Numerical solution of the quasi-neutrality equation	59
3.5.1	The decoupling approach	60
3.5.2	First approach: spectral + finite differences	61
3.5.3	Second approach: FEM	62
3.5.4	Numerical results	62
3.6	Conclusion	65

3.1 Introduction

Nowadays, the modeling of magnetized plasmas is a key issue for controlled thermonuclear fusion. In practice, the study of such plasmas requires solving the Maxwell equations coupled to the computation of the plasma response. Different ways are possible to compute this response: the fluid or the kinetic description. Obviously solving the full Vlasov equation involves the discretization of the six-dimensional phase space, which is a challenging problem. On the other hand, the fluid approach seems to be insufficient when one wants to study the behavior of zonal flow, or the interaction between waves and particles for example (see [56, 54]).

In the context of strongly magnetized plasmas however, the motion of the particles is particular since it is confined around the magnetic field lines; the frequency of this cyclotron motion is faster than the frequencies of interest. Hence, averaging the Vlasov equation over the cyclotron motion reduces the dimensionality, and numerical simulations, even if they remain very costly, become possible (see [51, 57]). These simulations are performed using particles methods or a phase space grid (Eulerian methods) but the computation of the electric potential is always performed on a physical 3D grid. Moreover, the configuration of a tokamak is such that the physics is highly anisotropic and structures along the magnetic field lines are quite larger than across the magnetic field lines. In order to reduce the numerical effort which is huge anyway, it is quite important to use this information in order to define the grid resolution in each direction. Indeed, if the grid is aligned or almost aligned with the magnetic field lines, it is possible to use a lot fewer points in the parallel direction than in the transverse direction.

It is of great importance to develop a quasi-neutrality solver that is flexible with respect to geometry and that can provide high order accuracy. In this work, we are investigating an approach which can accommodate arbitrary coordinates and a complicated geometry of the computation domain. In [27] Czarny and Huysmans used Bézier elements for MHD simulations. Bézier surfaces are the most basic tool in (CAD) computer aided design. However, it suffers from some disadvantages (preserving the exact geometry for a class of domains). The Isogeometric Analysis (IGA), which has been introduced recently by Hughes et al. [67], seems to provide all these features. The IGA relies on NURBS functions, which are a generalization of Spline functions and provides an exact modeling of large classes of computational domains including conics and all spline surfaces. Moreover they rely on a cartesian grid of the parameter space and are fairly easy to use even using spline basis functions of arbitrary degree. Moreover for domains that can be represented using a periodic angular variable as is the case for the poloidal plane of the tokamak, we were able to develop a fast solver that is comparable in computation time, for any spline degree, with the spectral and Finite Difference solver used in Gysela [51].

The results presented in this work use a bean shaped domain corresponding to a poloidal cut of the tokamak plasma. The next step will be to couple this field solver with a gyrokinetic solver. Moreover, being based on the projection of the approximated function into a finite dimensional space, it can not only deal with differential operators but also integral operators such as those involved in an exact computation of the double gyroaverage, which is up to now approximated by the transverse Laplacian.

Finally, let us mention a point which is particularly useful for parallel computations. Due to the adiabatic assumption for electrons, a nonlocal term intervenes in the equation which is very penalizing for massive parallelization of full gyrokinetic codes. In this

3.2. Quasi-neutrality equation

work, we propose an algorithm which enables an interesting decoupling of the quasi-neutrality equation. Indeed, by decomposing the electric potential between its average on a magnetic surface and the difference between itself and this average, it is possible to solve the quasi-neutrality equation by: first, solving a 1D radial ordinary differential equation, and second, solving N_φ poloidal 2D equations (where N_φ is the number of poloidal planes). This latter equations are solved using the NURBS approach. The two equations are local and seem suitable for massively parallel computations.

3.2 Quasi-neutrality equation

The Poisson equation enables to determine the electric potential ϕ as a function of the distribution function

$$\nabla^2 \phi = -4\pi |e| (n_i - n_e).$$

where n_i (resp. n_e) stands for the ion density (resp. the electron density). In classical tokamak plasmas, the Debye length is one order smaller than the Larmor radius so that (see [80]), the quasi-neutrality equation is given by

$$n_i(x) = n_e(x), \quad (3.2.1)$$

The ion density n_i is evaluated at particles position, and can be computed from the solution of the gyrokinetic equation (posed on guiding-center position). At the first gyrokinetic order, n_i can be written

$$n_i(x) = \int \mathcal{J}(f+g)(x, v) dv, \quad \mathcal{J}(f)(x) = \frac{1}{2\pi} \int_0^{2\pi} f(x+\rho) d\varphi, \quad \rho = |\rho|(\cos \varphi, \sin \varphi), \quad (3.2.2)$$

where f stands for the gyrocenter distribution and g is the first order correction, which can be approximated (as in [80, 116, 55]) by

$$g = \partial_\mu F_M(\phi - \mathcal{J}(\phi)), \quad \text{with } F_M = \frac{n_0}{\sqrt{2\pi T}} \exp(-m\mu/(2T)),$$

where $n_0 = n_0(r)$ is an equilibrium density and $T = T(r)$ is an equilibrium ion temperature. These two profiles are generally given. Hence, injecting this last expression into (3.2.2) leads to

$$n_i(x) = \bar{n}_i(x) + \frac{n_0}{T}(\phi(x) - \tilde{\phi}(x)), \quad (3.2.3)$$

where $\bar{n}_i(x) = \int \mathcal{J}(f)(x, v) dv$ and $\tilde{\phi}(x) = \int_0^\infty \mathcal{J}^2(\phi)(x) \exp(-\mu) d\mu$ is the second gyroaverage transformation of ϕ . In Fourier variables, this term has a compact form using the function $\Gamma_0(b) = \int_0^{+\infty} \exp(-x) \mathcal{J}_0^2(bx) dx$, where \mathcal{J}_0 is the Bessel function. Then (3.2.3) becomes in Fourier variables

$$\hat{n}_i = \hat{\bar{n}}_i + n_0 \frac{\hat{\phi}}{T_i} (1 - \Gamma_0(b)),$$

with $b = k_\perp^2 \rho_i^2$, $\rho_i^2 = T/B^2$. By expanding the Gamma function Γ_0 using a Padé approximation (see [34, 55, 116]), we obtain

$$n_i = \bar{n}_i + \frac{1}{B^2} \nabla_\perp \cdot (n_i \nabla_\perp \phi).$$

3.3. A fast solver for polar coordinates

For the electron density, an adiabatic assumption is often performed so that we can write the following equality

$$n_e(x) = n_0 + \frac{n_0}{T_e}(\phi - \langle \phi \rangle),$$

where the operator $\langle \cdot \rangle$ is an integration over constant magnetic surfaces. The quasi-neutrality equation then reads,

$$-\frac{1}{B^2} \nabla_{\perp} \cdot (n_i \nabla_{\perp} \phi) + \frac{n_0}{T_e} (\phi - \langle \phi \rangle) = \bar{n}_i - n_0.$$

Linearization around the equilibrium density n_0 together with the approximation $B \approx B_0$ are usually performed (see [55, 51]); these simplifications provides only a radial dependence for the anisotropic factor

$$-\frac{1}{B_0^2} \nabla_{\perp} \cdot (n_0 \nabla_{\perp} \phi) + \frac{n_0}{T_e} (\phi - \langle \phi \rangle) = \bar{n}_i - n_0. \quad (3.2.4)$$

In the rest of the paper, the equation (3.2.4) is intended to be solved.

3.3 A fast solver for polar coordinates

Let us now see how the specific structure of our problem allows us to derive a specific solver which is much faster than using a generic sparse matrix solver. For our quasi-neutrality equation, the mapping \mathbf{F} is the mapping defining polar coordinates $\mathbf{F}(r, \theta) = (r \cos \theta, r \sin \theta)$. Then we have $\det(J_{\mathbf{F}}) = r$. Moreover the matrix A is of the form $a(r)\mathbb{I}$ and $c \equiv c(r)$ is only a function of r . The matrix Θ becomes

$$\Theta = \begin{pmatrix} a(r) & 0 \\ 0 & \frac{a(r)}{r^2} \end{pmatrix}$$

On the other hand the basis functions can be written as products of functions of r only and of θ only. Let us also introduce a numbering using the two indices of our logical patch \tilde{Q} in (r, θ) . Thus the degree of freedom with index i will be associated with the grid index (i_r, i_{θ}) and the basis function R_i is such that $\tilde{R}_i(r, \theta) = \tilde{R}_{i_r}(r) \tilde{R}_{i_{\theta}}(\theta)$. Plugging this into the element integrals defining the stiffness and mass matrices, we get

$$\begin{aligned} & \int_{\tilde{Q}} \tilde{\nabla} \tilde{R}_i^T \Theta \tilde{\nabla} \tilde{R}_j \det(J_{\mathbf{F}}) d\xi d\eta + \int_{\tilde{Q}} \tilde{c} \tilde{R}_i \tilde{R}_j \det(J_{\mathbf{F}}) d\xi d\eta \\ &= \int_{\tilde{Q}} \left(a(r) \frac{\partial \tilde{R}_i}{\partial r} \frac{\partial \tilde{R}_j}{\partial r} + \frac{a(r)}{r^2} \frac{\partial \tilde{R}_i}{\partial \theta} \frac{\partial \tilde{R}_j}{\partial \theta} \right) r dr d\theta + \int_{\tilde{Q}} \tilde{c}(r) \tilde{R}_i \tilde{R}_j r dr d\theta \\ &= \int a(r) \tilde{R}'_{i_r}(r) \tilde{R}'_{j_r}(r) r dr \int \tilde{R}_{i_{\theta}}(\theta) \tilde{R}_{j_{\theta}}(\theta) d\theta + \int \frac{a(r)}{r^2} \tilde{R}_{i_r}(r) \tilde{R}_{j_r}(r) r dr \int \tilde{R}'_{i_{\theta}}(\theta) \tilde{R}'_{j_{\theta}}(\theta) d\theta \\ & \quad + \int \tilde{c}(r) \tilde{R}_{i_r}(r) \tilde{R}_{j_r}(r) r dr \int \tilde{R}_{i_{\theta}}(\theta) \tilde{R}_{j_{\theta}}(\theta) d\theta. \quad (3.3.5) \end{aligned}$$

These formulas lead us to two observations. First the elementary matrices decouple into products of integrals in r and integrals in θ so that the final matrices can be written in a Kronecker product structure (explanations and applications of this structure can be

3.3. A fast solver for polar coordinates

found in the article by Van Loan [113] and references therein). The exploitation of this structure for developing fast solvers was developed in [37]. Second the matrices in θ are simple mass and stiffness matrices with no varying parameter inside, so that if the mesh is uniform in θ the matrix will be circulant.

One way to express the Kronecker product structure is to write the unknown degrees of freedom and the right-hand-side as matrices where the terms correspond to the indices (i_r, i_θ) . We denote those respectively by U and F . Then, in our case, the linear system can be written

$$K_{ar}UM_\theta + M_{ar}UK_\theta + M_{cr}UM_\theta = F, \quad (3.3.6)$$

where K_{ar} is the weighted stiffness matrix in r corresponding to the terms $\int a(r)\tilde{R}'_{i_r}(r)\tilde{R}'_{j_r}(r)r dr$, M_{ar} is the weighted mass matrix in r corresponding to the terms $\int \frac{a(r)}{r^2}\tilde{R}_{i_r}(r)\tilde{R}_{j_r}(r)r dr$, M_{cr} is the weighted mass matrix in r corresponding to the terms $\int c(r)\tilde{R}_{i_r}(r)\tilde{R}_{j_r}(r)r dr$, K_θ is the stiffness matrix in θ corresponding to the terms $\int \tilde{R}'_{i_\theta}(\theta)\tilde{R}'_{j_\theta}(\theta)d\theta$ and M_θ is the mass matrix in θ corresponding to the terms $\int \tilde{R}_{i_\theta}(\theta)\tilde{R}_{j_\theta}(\theta)d\theta$. The latter two matrices K_θ and M_θ are circulant, which means that they can be both diagonalized in the same orthonormal basis corresponding to the Fourier modes. This can be expressed by

$$M_\theta = P\Lambda_M P^*, \quad K_\theta = P\Lambda_K P^*,$$

where Λ_M and Λ_K are the diagonal matrices of the eigenvalues and a multiplication by P corresponds to the normalized Fast Fourier Transform and a multiplication by P^* to its inverse.

This can be exploited for the fast solution of (3.3.6) bringing the solution of a linear system arising from a 2D problem to sets of smaller systems corresponding to 1D problems. The procedure can be performed with the following algorithm:

1. Multiply system (3.3.6) on the right by P (amounts to a 1D FFT on each line of F). Then we get

$$K_{ar}UP\Lambda_M + M_{ar}UP\Lambda_K + M_{cr}UP\Lambda_M = FP. \quad (3.3.7)$$

2. Note that a multiplication on the right by the diagonal matrix of eigenvalues corresponds to multiplying each column of the matrix by the corresponding eigenvalue, which implies that (3.3.7) corresponds to uncoupled problems on each of the columns of $\hat{U} = UP$. So denoting by $\hat{U}_1, \dots, \hat{U}_n$ the columns of the matrix \hat{U} and by $\hat{F}_1, \dots, \hat{F}_n$ the columns of the matrix $\hat{F} = FP$, (3.3.7) becomes for each column j ,

$$(\lambda_{M_j}(K_{ar} + M_{cr}) + \lambda_{K_j}M_{ar})\hat{U}_j = \hat{F}_j \quad (3.3.8)$$

This is a set of banded systems of size the number of points in the r direction, that can be solved very efficiently using the LAPACK routines DPBTRF for the Cholesky factorization that is only called once at the beginning and then DPBTRS for the solution at each time step.

3. Compute $U = \hat{U}P^*$ by inverse FFT of the lines of \hat{U} .

Let us now compute the cost at each time step for a $N_r \times N_\theta$ mesh, disregarding the cost of the Cholesky factorization for the systems in r which needs only to be performed once for a many time steps computation. The algorithm consists of three steps: 1) N_r

3.4. Numerical validation

FFTs which need $O(N_\theta \log_2 N_\theta)$ operations, 2) N_θ up and down sweeps of a Cholesky decomposed banded system which cost $O(N_r)$ each, 3) N_r inverse FFTs which cost $O(N_\theta \log_2 N_\theta)$ operations. So all together the cost is $O(N_r N_\theta \log_2 N_\theta)$ operations, which is almost optimal. This algorithm uses the structure of the system in an optimal manner and only works on dense matrices. A generic sparse systems solvers could not do this.

Numerical results

We have tested this new approach to solve an elliptic partial differential equation, using the analytic solution $u(r, \theta) = \sin(2\pi r) \sin(2\pi\theta)$, which solves :

$$-\nabla^2 u(r, \theta) + u(r, \theta) = F(r, \theta), \quad \Omega \quad (3.3.9)$$

where $\Omega = [0, 1] \times [0, 1]$, ∇^2 is the Laplacian cartesian. The boundary conditions are :

$$u(r = 0, \cdot) = u(r = 1, \cdot) = 0 \quad (3.3.10)$$

and periodic boundary condition on θ .

In Figures 3.1 and 3.2, we compare the time CPU spent to solve the linear system, using this approach, *spsolve*, from *scipy* based on SUPERLU solver, for the classical formulation (detailed in the sections before). The test was done on grids 128×128 and 256×256 , using different spline order. As one can see, the new method only slightly depends on the spline degree, the reason is that, for $1D$ problem, non-zeros elements are concentrated around the diagonal compared to the $2D$ case. On a grid of 512×512 , we spent 0.3 *sec* to solve the linear system, using cubic splines. Using splines of order 8, it took only 0.321 *sec*.

Spline degree	FIGA	SPLU	Spline degree	FIGA	SPLU
1	0.012	0.013	1	0.008	0.38
2	0.014	0.046	2	0.013	3.81
3	0.014	0.073	3	0.012	10.69
4	0.013	0.098	4	0.016	19.17
5	0.015	0.124	5	0.017	31.95
6	0.015	0.152	6	0.020	47.01
7	0.015	0.179	7	0.023	65.00

Figure 3.1: CPU-time, in seconds, spent in solving (left) and initializing (right) the linear system, using the new approach, namely Fast IGA, compared to SuperLU. Test done on a grid 128×128

3.4 Numerical validation

In all tests we have treated, we consider an elliptic problem under Dirichlet boundary condition, *i.e* $u(\mathbf{x}) = 0, \quad \forall \mathbf{x} \in \partial\Omega$.

3.4.1 Test case 1: Order of convergence for Poisson in polar coordinates

For the validation, we solved the polar coordinate elliptic problem, which fits into our generic elliptic problem (2.6.12) with A the identity tensor and $c = 0$,

$$-\nabla_{r,\theta}^2 \phi(r, \theta) = n(r, \theta),$$

3.4. Numerical validation

Spline degree	FIGA	SPLU	Spline degree	FIGA	SPLU
1	0.074	0.067	1	0.021	3.38
2	0.076	0.967	2	0.043	31.40
3	0.075	3.505	3	0.052	197.31
4	0.075	16.070	4	0.060	330.28
5	0.077	32.852	5	0.069	415.63

Figure 3.2: CPU-time, in seconds, spent in solving (left) and initializing (right) the linear system, using the new approach, namely Fast IGA, compared to SuperLU. Test done on a grid 256×256

with the following analytical solution:

$$\phi(r, \theta) = \sin \left(\frac{\pi}{r_{\max}^2 - r_{\min}^2} (r^2 - r_{\min}^2) \right).$$

Injecting the solution into the elliptic problem enables to compute the right hand side n which is given in input of the code. This test enables to check the L^2 norm of the error (in log scale) between the numerical and analytical solution, with respect to the parameter $h = \max\{\text{diam}(\tilde{Q})\}$, for different orders of the basis functions (from order 2 to order 6). This is shown in Figure 3.3. We verify that the slopes of the different curves correspond to the order of the basis functions. In particular, for high orders, the machine precision is achieved. We also plot in Figure 3.3 the CPU time as a function of the parameter $h = \max\{\text{diam}(\tilde{Q})\}$, for different orders of the basis functions (from order 2 to order 6). These two last figures gives information for choosing *a priori* the best compromise between precision (which order of the basis function should be chosen) and the CPU time.

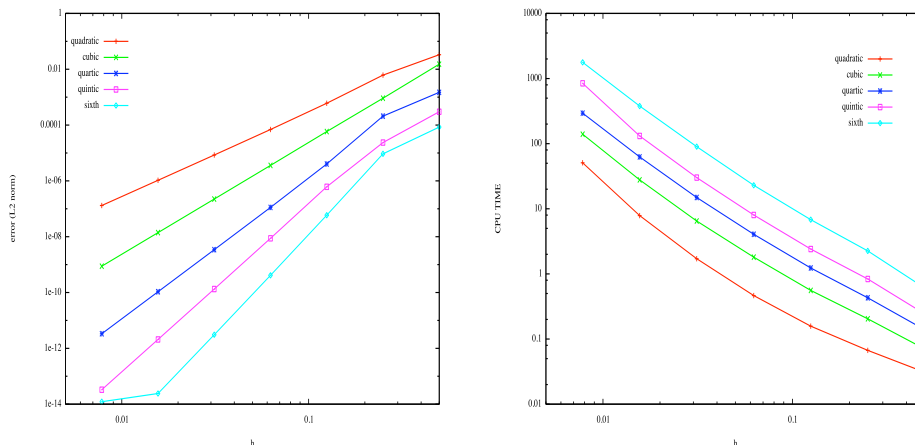


Figure 3.3: Validation test : (left) L^2 error norm, (right) CPU time.

3.4. Numerical validation

3.4.2 Test case 2: Chaotic solution

Following [88], we first test our solver on a chaotic solution on a polar coordinate Laplacian $-\nabla^2\phi = n$. This analytic solution takes the form

$$\phi_{\text{math}}(r, \theta) = \left[\sin(2\pi\xi) + \epsilon \sum_M B_M \sin(2\pi M\xi) \right] \sum_l A_l \cos(l\theta + \Theta_l), \quad (3.4.11)$$

where $0 \leq \epsilon \leq 1$, $\xi = (r - r_{\min})/(r_{\max} - r_{\min})$, A_l and B_M are random numbers which range between 0 and 1, with $|M|, |l| (\leq 20$ for the first simulation and ≤ 40 for the second one). Finally, the phase Θ_l is also given by a random number in $0, 2\pi$. The right hand side of $-\nabla^2\phi = n$ is given by

$$\begin{aligned} n(r, \theta) &= -\frac{1}{r} (\partial_r(r\partial\phi_{\text{math}})) - \frac{1}{r^2} \partial_\theta^2 \phi_{\text{math}} \\ &= -\frac{1}{r} \sigma_r(r, \theta) \sum_l A_l \cos(l\theta + \Theta_l) \\ &\quad + \left[\sin(2\pi\xi) + \epsilon \sum_M B_M \sin(2\pi M\xi) \right] \sum_l \frac{l^2}{r^2} A_l \cos(l\theta + \Theta_l), \end{aligned}$$

with

$$\begin{aligned} \sigma_r(r, \theta) &= \frac{2\pi}{\Delta r} \cos(2\pi\xi) - \frac{4\pi^2 r}{\Delta r^2} \sin(2\pi\xi) \\ &\quad + \epsilon \sum_M B_M \left[\frac{2\pi M}{\Delta r} \cos(2\pi M\xi) - \frac{4\pi^2 M^2 r}{\Delta r^2} \sin(2\pi M\xi) \right] \end{aligned}$$

In our numerical experiments, ϵ is taken equal to 0.4. Note that the solution satisfies the homogeneous Dirichlet condition at $r = r_{\min} = 0.2$ and $r = r_{\max} = 0.4$ and periodic boundary conditions in the θ direction.

We show on Figure 3.4, the L^2 norms (in log scale) of the difference between the analytical and numerical solutions as a function of the parameter $h = \max\{\text{diam}(\tilde{Q})\}$. The same observations as before are also available for this test; the slopes of the curves corresponds to the order of the basis, even when a very large number of modes is considered in the solution. In this kind of test high order basis functions enable to better capture the very fine scales of the solution.

Spline degree	Degrees of freedom	L^2 error norm
2	17408	$7.20 \cdot 10^{-3}$
3	18060	$1.07 \cdot 10^{-3}$
4	18720	$1.71 \cdot 10^{-4}$
5	19388	$2.84 \cdot 10^{-5}$
6	20064	$4.81 \cdot 10^{-6}$
7	20748	$1.67 \cdot 10^{-6}$

Table 3.1: *Nishimura test:* Number of degree of freedom and L^2 norm of the error for each spline degree.

Comparing with Nishimura results [88], we see in Table 3.1 that the new method allows us to reach same error order, with much fewer degrees of freedom, by increasing

3.5. Numerical solution of the quasi-neutrality equation

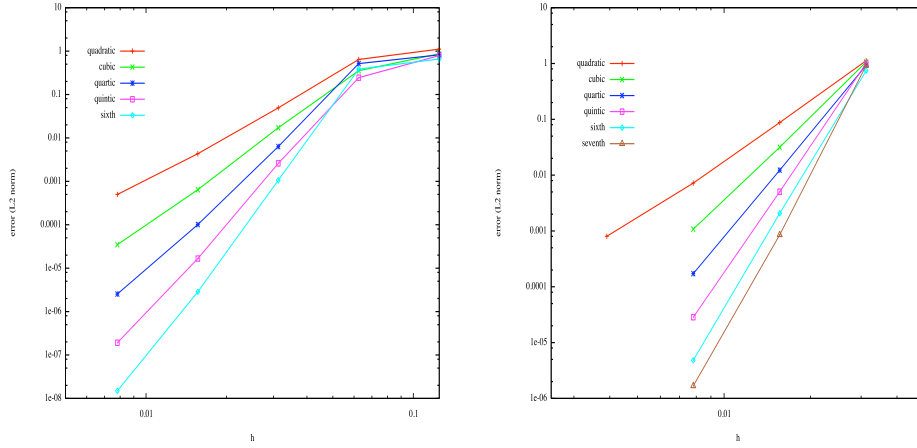


Figure 3.4: Nishimura test: L^2 error norm as a function of h ; (left) for 20 modes, (right) 40 modes.

the spline order. Let us notice, that to reach an error of 10^{-6} , Nishimura used a grid of $N = 154,560$, with our method, we only need 8 times less ($N = 20064$) degrees of freedom by using sixth or seventh order for spline basis.

3.5 Numerical solution of the quasi-neutrality equation

This section is devoted to the numerical approximation of the quasi-neutrality equation (3.2.4) by applying the method introduced above. To that purpose, we will consider a given right hand side (that is to say the distribution function of the ions is given and the coupling with the gyrokinetic equation is not considered), and we focus on the computation of the solution of (3.2.4) in polar coordinates:

Given $n_0(r), T_e(r)$ two radial profiles and $F(r, \theta, \varphi)$, solve for $\phi(r, \theta, \varphi)$ satisfying

$$-\nabla_{\perp} \cdot (n_0 \nabla_{\perp} \phi) + \frac{n_0}{T_e} (\phi - \langle \phi \rangle) = F(r, \theta, \varphi). \quad (3.5.12)$$

The $\langle \cdot \rangle$ operator refers to the magnetic flux average

$$\langle \phi \rangle(t, r) = \frac{1}{\int \int J(r, \theta) d\theta d\varphi} \int \int \phi(t, r, \theta, \varphi) J(r, \theta) d\theta d\varphi, \quad (3.5.13)$$

with $J(r, \theta)$ a jacobian defining the poloidal geometry, whereas the given right hand side $F(r, \theta, \varphi)$ reads

$$F(r, \theta, \varphi) = (\bar{n}_i(r, \theta, \varphi) - n_0(r)),$$

where \bar{n}_i is the ion density, T_e the electronic temperature and n_0 is a correction term taking into account the presence of the electrons. The first term on the left hand side is known as the polarization term which corresponds to the difference between the guiding-center density and that of particles.

The boundary conditions are supposed to be 2π -periodic in the θ, φ variables, whereas the radial boundary conditions are imposed by the Dirichlet condition and writes $\phi(r = r_{\min}, \theta, \varphi) = 0$ and $\phi(r = r_{\max}, \theta, \varphi) = 0$.

3.5. Numerical solution of the quasi-neutrality equation

The main goal of this section consists in the numerical solution of (3.5.12). More precisely, we want to derive an efficient method for (3.5.12), that is to say a method which is as local as possible. Indeed, the principal difficulty is the average term $\langle \phi \rangle$ which is totally nonlocal since it couples every values of θ and φ . The nonlocality is an important obstacle for an efficient parallelization, but also when one wants to use Fourier transforms in the θ, φ variables. We propose a decoupling approach that enables to decompose the solution of (3.5.12) into two local problems, one of them reduces to a two-dimensional elliptic type problem of the form (2.6.12).

The rest of this section presents different ways to solve the quasi-neutrality equation, pointing out the advantages and disagreements of the solvers. Then, some numerical results are shown to compare the performance of the solvers.

3.5.1 The decoupling approach

In this subsection, we propose a new method to allow us the derivation of a local algorithm for (3.5.12) in the φ variable. An efficient algorithm can then be developed for its resolution. To this purpose, we first restrict in this section to a Jacobian such that $J(r, \theta) = r$, but the computations for arbitrary Jacobians $J(r, \theta)$ are performed in Appendix A. The main ingredient consists in the decoupling of (3.5.12) into two local equations: one equation on $\langle \phi \rangle$ which is given by (3.5.13), and one equation on $\Phi = \phi - \langle \phi \rangle$.

The first equation is simply derived by averaging the left hand side of the quasi-neutrality equation (3.5.12) with respect to θ and φ variables

$$\frac{1}{(2\pi)^2} \int \int \nabla_{\perp} \cdot [n_0(r) \nabla_{\perp} \phi] d\theta d\varphi = \left[\left(\frac{n_0(r)}{r} + n'_0(r) \right) \partial_r \langle \phi \rangle + n_0(r) \partial_r^2 \langle \phi \rangle \right]. \quad (3.5.14)$$

Finally, introducing $\langle F \rangle(r) = 1/(2\pi)^2 \int \int F(r, \theta, \varphi) d\theta d\varphi$, we have

$$\langle \nabla_{\perp} \cdot [n_0(r) \nabla_{\perp} \phi] \rangle = \nabla_{\perp} \cdot [n_0(r) \nabla_{\perp} \langle \phi \rangle] = \langle F \rangle, \quad (3.5.15)$$

from which we can deduce a 1D equation for $\langle \phi \rangle = \langle \phi \rangle(r)$

$$- \left[n_0(r) \partial_r^2 \langle \phi \rangle + \left(\frac{n_0(r)}{r} + n'_0(r) \right) \partial_r \langle \phi \rangle \right] = \langle F \rangle(r), \quad (3.5.16)$$

Then, we want to derive an equation satisfied by $\Phi = \phi - \langle \phi \rangle$. This can be done easily by adding and removing the term $\nabla_{\perp} \cdot [n_0(r) \nabla_{\perp} \langle \phi \rangle]$ in (3.5.12)

$$-\nabla_{\perp} \cdot [n_0(r) \nabla_{\perp} (\phi - \langle \phi \rangle)] + \frac{n_0}{T_e} [\phi - \langle \phi \rangle] - \nabla_{\perp} \cdot [n_0(r) \nabla_{\perp} \langle \phi \rangle] = F(r, \theta, \varphi).$$

By introducing the new unknown $\Phi = \phi - \langle \phi \rangle$ in this last equation, we get

$$-\nabla_{\perp} \cdot [n_0(r) \nabla_{\perp} \Phi] + \frac{n_0}{T_e} \Phi - \nabla_{\perp} \cdot [n_0(r) \nabla_{\perp} \langle \phi \rangle] = F(r, \theta, \varphi). \quad (3.5.17)$$

Thanks to (3.5.15), we can write the equation which is satisfied by $\Phi(r, \theta, \phi) = \phi - \langle \phi \rangle$

$$-\nabla_{\perp} \cdot [n_0(r) \nabla_{\perp} \Phi] + \frac{n_0}{T_e} \Phi = F(r, \theta, \varphi) - \langle F \rangle(r). \quad (3.5.18)$$

More precisely, the equation to solve is

$$-n_0 \partial_r^2 \Phi - \left[\frac{n_0}{r} + n'_0 \right] \partial_r \Phi - \frac{n_0}{r^2} \partial_{\theta}^2 \Phi + \frac{n_0}{T_e} \Phi = F(r, \theta, \varphi) - \langle F \rangle(r). \quad (3.5.19)$$

3.5. Numerical solution of the quasi-neutrality equation

Thanks to these straightforward computations, we totally decoupled (3.5.12) by introducing the new unknowns Φ and $\langle \phi \rangle$. Moreover, equation (3.5.12) is strictly equivalent to the equations (3.5.16)-(3.5.18). This new formulation provides a system of equation which does not include nonlocal term any more. The algorithm is then the following Algorithm

- solve the 1D equation (3.5.16) to get $\langle \phi \rangle(r)$
- solve the 2D equation (3.5.18) $\forall \varphi$ to get $\Phi(r, \theta, \varphi)$
- compute $\phi = \Phi + \langle \phi \rangle$

Now, an important point consists in a good choice of the numerical approximation to solve the 2 equations (3.5.16) and (3.5.18). A two-dimensional solver will provide Φ whereas a one-dimensional solver for (3.5.16) leads to $\langle \phi \rangle$. By the summation of these two solutions, we can obtain the desired solution $\phi(r, \theta, \varphi)$.

3.5.2 First approach: spectral + finite differences

The first approach to solve (3.5.18) consists in the use of a Fourier transform in the θ direction (Φ is periodic in the θ variable since ϕ is). The projection of $\Phi(r, \theta, \varphi)$ onto the Fourier space writes

$$\Phi(r, \theta, \varphi) = \sum_{m=1}^N \Phi^m(r, \varphi) e^{im\theta}.$$

so that equation (3.5.19) is rewritten

$$-\left(n_0 \partial_r^2 \Phi^m + \left(\frac{n_0}{r} + n_0'(r) \right) \partial_r \Phi^m \right) + n_0 \left(\frac{1}{T_e} + \frac{m^2}{r^2} \right) \Phi^m = S^m(r, \phi), \quad (3.5.20)$$

where

$$S^m(r, \phi) = \begin{cases} F^m(r, \varphi) & \text{if } m \neq 0 \\ F^0(r, \varphi) - \frac{1}{(2\pi)^2} \int F^0(r, \varphi) d\varphi & \text{else,} \end{cases}$$

is the Fourier coefficient of order m of the right hand side of (3.5.18). We are faced to an ODE for each Fourier mode m ; one classical way to solve this problem is the use of a finite difference scheme in the r direction to solve (3.5.20); this leads to the constitution of a tridiagonal linear system, the resolution of which allows to get the Fourier modes $\Phi^m(r, \varphi)$. An inverse Fourier transform leads to $\Phi(r, \theta, \varphi)$.

The resolution of the ODE (3.5.16) can be performed in the same way as (3.5.20), using finite differences of order two.

Remark 3.5.1 *From the CPU time point of view, the present approach enables us to solve the total quasi-neutrality equation (i.e. to compute $\phi = \langle \phi \rangle + \Phi$) with a cost of order $O(N^2 \ln(N))$ for a $N \times N$ grid in (r, θ) . Our NURBS based solver has the same complexity with only the bandwidth of the systems in r that increase with the degree of the splines.*

3.5. Numerical solution of the quasi-neutrality equation

3.5.3 Second approach: FEM

The second option discretizes directly equation (3.5.18), that considers two dimensions (r, θ) and a parameter φ . The IGA finite element method is performed here.

For the ODE (3.5.16), a high order method is required not to penalize the high order achieved in the two-dimensional solution. Here, we propose a collocation method introduced in [19].

Recalling, that the equation 3.5.18 can be written in the form:

$$-\nabla_{\perp} \cdot (A \nabla_{\perp} \Phi) + c \Phi = g \quad (3.5.21)$$

where,

$$A = \begin{pmatrix} n_0 & 0 \\ 0 & n_0 \end{pmatrix}, \quad c = \frac{n_0}{T_e}, \quad \text{and,} \quad g = F(r, \theta, \varphi) - \langle F \rangle(r). \quad (3.5.22)$$

Remark 3.5.2 *A third approach could be the use of a spline FEM to solve the ODE. We can use the same 1D matrices that we have already assembled and appear in the Kronecker product.*

3.5.4 Numerical results

In the present section, we compare the different approximation of the quasi-neutrality equation (3.5.12) we proposed in the previous section. The different methods are studied on analytic tests, since we impose a right hand side which is consistent with a solution of (3.5.12).

Test case 1

In this test, we consider the full quasi-neutrality equation (3.5.12) in which the profile n_0 and T_e are supposed constant equal to 1, with the following solution

$$\phi(r, \theta, \varphi) = \sin\left(\frac{2\pi}{L}(r - r_{\min})\right) u(\theta, \varphi), \quad \theta, \varphi \in [0, 2\pi], r \in [0.2, 0.8],$$

with $L = r_{\max} - r_{\min}$ and $u(\theta, \varphi) = 1/\pi^2(\cos^2 \theta \cos^2 \varphi)$. This form of solution imposes the average to be $\langle \phi \rangle(r) = \sin\left(\frac{2\pi}{L}(r - r_{\min})\right)$. Injecting this solution in (3.5.12) leads to an analytical right hand side which is used to recover numerically the true solution. This analytical way enables comparison with the analytical solution. Let us detail the computations leading to the right hand side. First, we compute the three component of $\nabla_{\perp}^2 \phi$

$$\begin{aligned} -\partial_r^2 \phi &= \frac{4\pi^2}{L^2} \sin\left(\frac{2\pi}{L}(r - r_{\min})\right) u(\theta, \varphi), \\ -\frac{1}{r} \partial_r \phi &= \frac{2\pi}{L} \cos\left(\frac{2\pi}{L}(r - r_{\min})\right) u(\theta, \varphi), \\ -\frac{1}{r^2} \partial_{\theta}^2 \phi &= \frac{2}{r^2 \pi^2} \sin\left(\frac{2\pi}{L}(r - r_{\min})\right) \cos(2\theta) \cos^2 \varphi, \end{aligned}$$

Then, we can compute $\langle F \rangle$ which will be used in the right hand side of equations (3.5.16) and (3.5.18)

$$\langle F \rangle(r) = \frac{4\pi^2}{L^2} \sin\left(\frac{2\pi}{L}(r - r_{\min})\right) - \frac{2\pi}{rL} \cos\left(\frac{2\pi}{L}(r - r_{\min})\right).$$

3.5. Numerical solution of the quasi-neutrality equation

We also compute $F(r, \theta, \varphi)$ for the resolution of (3.5.18)

$$\begin{aligned}
 F(r, \theta, \varphi) &= \sin\left(\frac{2\pi}{L}(r - r_{\min})\right) u(\theta, \varphi) \left[\frac{4\pi^2}{L^2} + 1\right] \\
 &+ \frac{2\pi}{L} \cos\left(\frac{2\pi}{L}(r - r_{\min})\right) u(\theta, \varphi) \\
 &+ \frac{2}{r^2\pi^2} \sin\left(\frac{2\pi}{L}(r - r_{\min})\right) \cos(2\theta) \cos^2\varphi - \sin\left(\frac{2\pi}{L}(r - r_{\min})\right) \\
 &= \sin\left(\frac{2\pi}{L}(r - r_{\min})\right) u(\theta, \varphi) \left[\frac{4\pi^2}{L^2} + 1\right] \\
 &+ u(r, \theta) \left[\sin\left(\frac{2\pi}{L}(r - r_{\min})\right) - \frac{1}{r} \frac{2\pi}{L} \cos\left(\frac{2\pi}{L}(r - r_{\min})\right)\right] \\
 &- \langle F \rangle(r) - \sin\left(\frac{2\pi}{L}(r - r_{\min})\right)
 \end{aligned}$$

We then test the decoupling method for which the spectral approach, the IGA-FEM solution is compared to the analytic solution. For the first approach, we fix the number of points in the angular directions θ, φ to 64 whereas the number of points in the radial direction is modified to recover the order of the finite difference method.

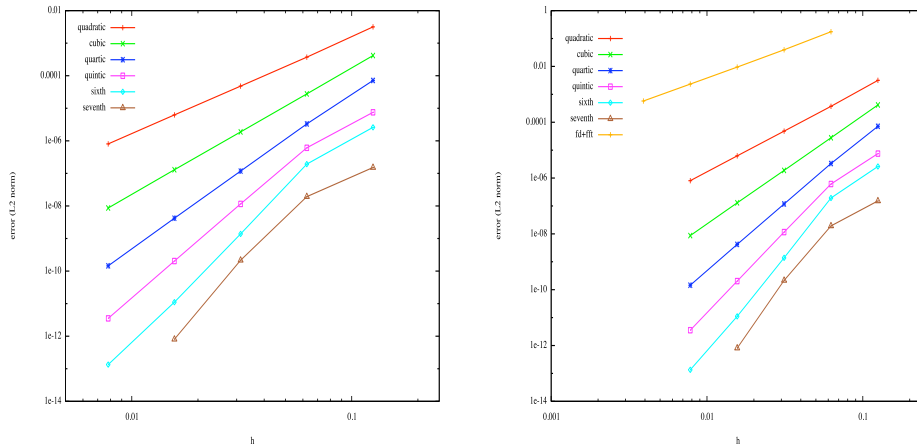


Figure 3.5: Test case 1: L^2 error norm, (left) for the elliptic part, (right) for the global problem ODE + FEM.

We show in the following figures comparisons between the different approaches we propose. On Figures 3.5, we plot the L^2 norm errors between the numerical and analytical solution. We first plot the error for the elliptic part (3.5.18) for which the behavior is similar to the curves obtained in chapter 2. Then, we focus on the total error of the solvers for (3.5.12). We observe that the curves are nearly the same; this is due to the collocation method which is very precise. The error made in this part is then negligible compared to the error made in the elliptic part. We can also remark that the error for the classical approach spectral + finite differences is of order 2 (due to the second order finite differences). This approach needs to consider a lot of point in the radial direction (2048) to reach a competitive accuracy, whereas the IGA-FEM approach presents very precise results even considering cubic or quartic order.

3.5. Numerical solution of the quasi-neutrality equation

Test case 2

In this test, some anisotropy is introduced in the Laplacian term by initializing n_0 . Moreover, a profile is imposed to T_e . These two functions are solutions of a differential equation. The radial profiles of the ion temperature $T_e(r)$ as well as the radial density profile $n_0(r)$ are deduced by numerical integration of their gradient profile given by

$$\frac{1}{T_e(r)} \frac{dT_e(r)}{dr} = -\cosh^{-2}(3(r - r_p)), \quad (3.5.23)$$

with $r_p = 0.5$. These parameters are employed in [51]. Hence, imposing analytical form of the solution,

$$\phi(r, \theta, \varphi) = C \sin \varphi \cos(4\theta) (r - r_{\min})^6 (r_{\max} - r)^6, \quad (3.5.24)$$

with C a constant chosen such that $\max[\phi] = 1$, we deduce an expression of the right hand side of (3.5.12) which is given to the code to compute back an approximation of the solution. The right hand side F is given by

$$F(r, \theta, \varphi) = -\partial_r^2 \phi - \partial_r \phi \left(\frac{1}{r} + \frac{n_0'}{n_0} \right) - \frac{1}{r^2} \partial_\theta^2 \phi + \frac{\phi}{T_e},$$

where the different term are

$$\begin{aligned} \partial_r \phi &= 6C \sin(4\theta) \sin \varphi [(r - r_{\min})^5 (r_{\max} - r)^6 - (r - r_{\min})^6 (r_{\max} - r)^5], \\ \partial_r^2 \phi &= 30C \sin(4\theta) \sin \varphi (r - r_{\min})^4 (r_{\max} - r)^4 \\ &\quad \left[(r_{\max} - r)^2 - \frac{12}{5} (r - r_{\min})(r_{\max} - r) + (r - r_{\min}) \right], \\ \partial_\theta^2 \phi &= -16\phi. \end{aligned}$$

As in the previous tests, we are interested in the L^2 norm of the error as a function of the size of the mesh. Results are presented in Table 3.2 in which the L^2 norm of the error is given as a function of the number of points and the degree of the splines. We also give the results for the standard approach where finite difference and FFT are used. In this test also, the IGA-FEM approach has a very good behaviour since with 32 points per direction; to achieve the same precision the finite difference+FFT approach needs 1024 points per direction. Note a kind of saturation of the error: this is due to the computation of the solution of the ODE 3.5.23. Indeed, T_e and n_0 are numerical solution of an ODE and are not analytical. We verify that when the ODE is solved precisely (by refinement), the correct orders are recovered.

On Figure 3.6, the solution together with the error between the analytical solution and the numerical one are plotted for the IGA-FEM approach, using degree 2 and 32 points per direction.

Test case 3: non-circular cross section

Following the Nishimura idea [88], we have generated a turbulence test over our non-circular tokamak model. We give the result of such test, with $l = m = 10$ and NURBS degree $p = 4$, in Figure 3.7.

For such a domain, the use of the standard finite elements method, would need a great number of degrees of freedom, moreover we couldn't be able to represent exactly the tokamak.

3.6. Conclusion

Spline degree	Degrees of freedom	L^2 error norm	L^2 error norm "refined"
FD+FFT	262144	$1.5 \cdot 10^{-5}$	
FD+FFT	1048576	$3.7 \cdot 10^{-6}$	
FD+FFT	4194304	$9.6 \cdot 10^{-7}$	
2	256	$4.5 \cdot 10^{-5}$	$4.5 \cdot 10^{-5}$
2	1024	$6.4 \cdot 10^{-6}$	$2.3 \cdot 10^{-6}$
3	256	$9.6 \cdot 10^{-6}$	$7.6 \cdot 10^{-6}$
3	1024	$7.5 \cdot 10^{-6}$	$5.7 \cdot 10^{-7}$
4	256	$5.3 \cdot 10^{-6}$	$1.6 \cdot 10^{-6}$
4	1024	$6.6 \cdot 10^{-6}$	$4.8 \cdot 10^{-7}$

Table 3.2: Test 2 QN: Number of degree of freedom and L^2 norm of the error for some spline degrees and for the method finite-difference+FFT (FD+FFT). In the third column, the ODE (3.5.23) is solved using 10^3 points whereas in the last column, 10^4 are used.

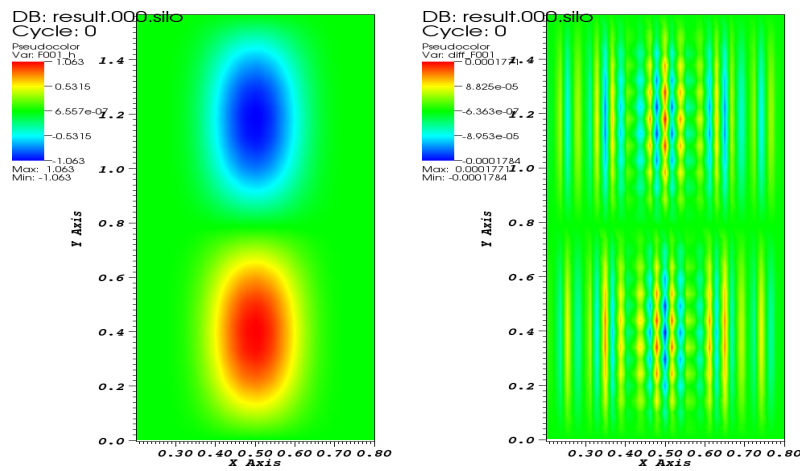


Figure 3.6: Test 2 QN: numerical solution with NURBS (left) and the difference with the analytical solution (right). 32 points are used per direction, order 3.

3.6 Conclusion

In this work we have developed an adequate solver of the Quasi-Neutral equation, using the IGA approach. Currently, we are trying to couple the Fast polar solver in the Gysela code. In the future, we hope that Gysela will be able to deal with complex geometries using the IGA, for this purpose we need to develop Semi-Lagrangian schemes using *B-splines/NURBS* mappings. This is the subject of the chapter 6.

3.6. Conclusion

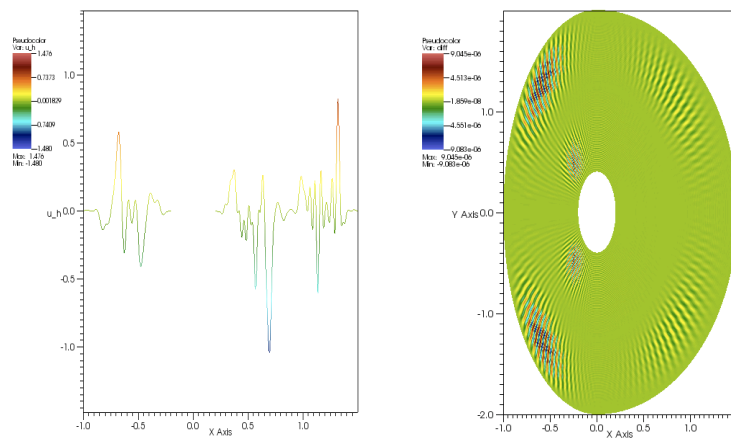


Figure 3.7: Turbulence test: (left) the numerical solution, with $l = m = 10$ and $p = 4$, (right) the difference $u - u_h$ for a turbulence test in the tokamak model, with $l = m = 2$ and $p = 4$.

Application to the 2D Maxwell's equations

Contents

4.1	Introduction	68
4.2	Variational formulation for the 2D Maxwell equations	69
4.3	Construction of the finite element spaces	70
4.3.1	Spline finite elements on patch grids	71
4.3.2	The Discrete Equations	72
4.4	Leap Frog scheme's stability	73
4.5	Numerical results	74
4.5.1	Test case 1: square	75
4.5.2	Test case 2: circular wave guide	77
4.5.3	Test case 3: Silver-Muller condition	78
4.6	H-rot formulation	79
4.7	Axisymmetric Variational formulation of the 2D Maxwell's equation .	80
4.7.1	Discrete equations - 1 st formulation	81
4.7.2	Discrete equations - 2 nd formulation	83
4.7.3	H-rot formulation	84
4.7.4	Remarks	85
4.8	Conclusions and perspectives	86

4.1 Introduction

Since their introduction, B-splines have had a great success, thanks mainly to fast and stable algorithms developed for their use. They are used as well in industry as in academic research for interpolation, data fitting and computer aided design. Recent works by Hughes and co-authors [67, 23] and the introduction of iso-geometric analysis have added yet another dimension to their use, creating an interface between simulation and numerical modeling.

It seems that before the recent works of Hughes [67], the use of splines as basis functions in the finite element method was quite uncommon and essentially limited to uniform B-splines using periodic conditions, even though the web-splines developed by Hoellig and co-workers provided a strategy for dealing with boundary conditions [63, 59]. The idea of iso-geometric analysis using geometric transformations and non uniform splines or NURBS appears much simpler for most applications. Compared to traditional finite elements, the main change due to the iso-geometric analysis is undoubtedly the emergence of the k -refinement, a strategy that can increase the regularity of functions through the mesh's interfaces, to reduce the number of degrees of freedom.

Modern finite element techniques for Maxwell's equations rely on ideas from differential geometry and more precisely on the existence of discrete spaces that provide an exact De Rham sequence. Following pioneering ideas by Bossavit [13, 14] a complete theory was successively developed [95, 85]. Buffa and co-workers [16, 18, 17] extended iso-geometric analysis to steady-state Maxwell's equations providing a discrete exact De Rham sequence involving discrete spaces based on B-splines. In [15], Buffa et al studied the 2D Stokes equation.

Our aim is to use this discrete sequence for the solution of the time-dependent Maxwell equations. One important feature of geometric Finite Maxwell solvers is that one of Faraday's or Ampere's law hold strongly in the discrete spaces and the other one needs a Galerkin Hodge operator [14]. Hence, one of the discrete equations is completely explicit and the other one involves a mass matrix that yields a linear system to be solved at each iteration. Thanks to a property of B-spline derivatives, we were able to exhibit basis functions of the discrete spaces involved in the De Rham sequence, such that the same property holds and the discrete curl or divergence, depending on the chosen formulation, is simply an incidence matrix depending only on the connectivity of the mesh, which is thus independent of the geometric transformation. We have implemented this idea in 2D, but as it is based on a tensor product approximation, it extends straightforwardly to 3D.

The outline of the paper is the following. First we recall the variational formulation of the 2D Maxwell equation, then we give the most important properties of B-splines and iso-geometric analysis. After that, we construct our exact sequence of discrete spaces as spans of well chosen basis functions on a cartesian grid and explain how to transform them on the geometric domain defined by NURBS. We can then compute the semi-discrete equations in space in a matrix formulation. We then prove a stability condition when a Leap-Frog algorithm in time is used. Finally, different test cases are performed to validate the method. High-order convergence is obtained and CFL conditions are computed for different degrees of splines.

4.2 Variational formulation for the 2D Maxwell equations

In the sequel, Ω will denote a subdomain in \mathbb{R}^2 , for which there exists a mapping F that maps a square onto Ω . Our theory is restricted to a class of functions F which are C^1 . The Computer Aided Design (CAD) gives us a simple way to construct or approach such functions, (using B-splines and NURBS for example). We will denote by $\Gamma = \partial\Omega$ the boundary of Ω , and \mathbf{n} , the outward unit normal vector of Ω on the boundary Γ .

We recall that in 2D, we have two curl operators, one acting on scalars $\mathbf{rot} u = \begin{pmatrix} \frac{\partial u}{\partial y} \\ -\frac{\partial u}{\partial x} \end{pmatrix}$, and one acting on vectors $\mathbf{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$ for which

$$\mathbf{rot} \mathbf{v} = \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}. \text{ The divergence of a vector } \mathbf{v} \text{ is defined by } \operatorname{div} \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}.$$

We shall also need the following function spaces

$$H(\operatorname{curl}, \Omega) = \{ \mathbf{v} \in (L^2(\Omega))^2; \operatorname{rot} \mathbf{v} \in L^2(\Omega) \} \text{ and } H_0(\operatorname{curl}, \Omega) = \{ \mathbf{v} \in H(\operatorname{curl}, \Omega); \mathbf{v} \times \mathbf{n} = 0 \},$$

$$H(\operatorname{div}, \Omega) = \{ \mathbf{v} \in (L^2(\Omega))^2; \operatorname{div} \mathbf{v} \in L^2(\Omega) \}.$$

Notice that the space that could be called

$$H(\mathbf{rot}, \Omega) = \{ u \in (L^2(\Omega))^2; \mathbf{rot} u \in L^2(\Omega) \}$$

is identical to $H^1(\Omega)$. We shall therefore stick with the more usual $H^1(\Omega)$.

Finally, we recall the Green formula we will need:

$$\int_{\Omega} (\mathbf{rot} G) \cdot \mathbf{F} dX = \int_{\Omega} G \operatorname{rot} \mathbf{F} dX - \int_{\Gamma} (G \times \mathbf{n}) \cdot \mathbf{F} dS, \forall \mathbf{F} \in H(\operatorname{curl}, \Omega), \forall G \in H^1(\Omega),$$

and

$$\int_{\Omega} (\operatorname{div} \mathbf{F}) G dX = - \int_{\Omega} \mathbf{F} \cdot (\nabla G) dX + \int_{\Gamma} \mathbf{F} \cdot \mathbf{n} G dS, \forall \mathbf{F} \in H(\operatorname{div}, \Omega), \forall G \in H^1(\Omega). \quad (4.2.1)$$

In 2D domains, Maxwell's equations can be decoupled into two systems. The first involving the (E_x, E_y, H_z) components is called the Transverse Electric (TE) mode, and the second, involving the (H_x, H_y, E_z) components is called the Transverse Magnetic (TM) mode. As both modes can be discretized in the same manner, we shall only consider in the sequel the TE mode which reads

$$\frac{\partial \mathbf{E}}{\partial t} - \mathbf{rot} H = -\mathbf{J}, \quad (4.2.2)$$

$$\frac{\partial H}{\partial t} + \operatorname{rot} \mathbf{E} = 0, \quad (4.2.3)$$

$$\operatorname{div} \mathbf{E} = \rho, \quad (4.2.4)$$

where the components are defined by $\mathbf{E} = \begin{pmatrix} E_x \\ E_y \end{pmatrix}$, $H = H_z$. These equations need to be supplemented with initial and boundary conditions. We shall only consider periodic or perfectly conducting boundary conditions $\mathbf{E} \times \mathbf{n} = 0$ and in a second step Silver-Müller absorbing boundary conditions.

In order to derive a conforming Finite Element approximation of Maxwell's equations we first need to write an appropriate variational formulation. We would like to stay with the first order version of the system and are then naturally led to a mixed formulation

4.3. Construction of the finite element spaces

involving two different functional spaces for \mathbf{E} and H . The two options are, after multiplying both equations by a test function and integrating by parts, to use Green's formula for either one of the two equations but not for both.

The first variational formulation, in the case of perfectly conduction boundary conditions, can be derived using Green's formula (4.7.21) in Ampere's law (4.2.2). This yields *Find* $(\mathbf{E}, H) \in H_0(\text{curl}, \Omega) \times L^2(\Omega)$ such that

$$\frac{d}{dt} \int_{\Omega} \mathbf{E} \cdot \boldsymbol{\psi} \, dX - \int_{\Omega} H(\text{rot } \boldsymbol{\psi}) \, dX = - \int_{\Omega} \mathbf{J} \cdot \boldsymbol{\psi} \, dX \quad \forall \boldsymbol{\psi} \in H_0(\text{curl}, \Omega), \quad (4.2.5)$$

$$\frac{d}{dt} \int_{\Omega} H\varphi \, dX + \int_{\Omega} (\text{rot } \mathbf{E})\varphi \, dX = 0 \quad \forall \varphi \in L^2(\Omega). \quad (4.2.6)$$

Using the Green formula (4.7.21) in Faraday's law (4.2.3) yields the second variational formulation

Find $(\mathbf{E}, H) \in H(\text{div}, \Omega) \times H^1(\Omega)$ such that

$$\frac{d}{dt} \int_{\Omega} \mathbf{E} \cdot \boldsymbol{\psi} \, dX - \int_{\Omega} (\text{rot } H) \cdot \boldsymbol{\psi} \, dX = - \int_{\Omega} \mathbf{J} \cdot \boldsymbol{\psi} \, dX \quad \forall \boldsymbol{\psi} \in H(\text{div}, \Omega), \quad (4.2.7)$$

$$\frac{d}{dt} \int_{\Omega} H\varphi \, dX + \int_{\Omega} \mathbf{E} \cdot (\text{rot } \varphi) \, dX = 0 \quad \forall \varphi \in H^1(\Omega). \quad (4.2.8)$$

Notice that in the first variational (4.2.5)-(4.2.6) formulation the boundary condition is taken into account in strong form by putting it into the function space where \mathbf{E} lives. On the other hand, in the second formulation (4.2.7)-(4.2.8) the boundary condition is taken into account in a weak form.

4.3 Construction of the finite element spaces

An important feature of the functional spaces we chose for the variational formulation is that they form an exact sequence. Depending of the variational formulation we choose, we need to work with different exact sequences. In the case of (4.2.5)-(4.2.6), the following function spaces are involved

$$\begin{array}{ccccc} & \mathbf{grad} & & \mathbf{rot} & \\ H^1(\Omega) & \longrightarrow & H(\text{curl}, \Omega) & \longrightarrow & L^2(\Omega) \\ \cup & & \cup & & \cup \\ V & \longrightarrow & W_{\text{curl}} & \longrightarrow & X \end{array} \quad (4.3.9)$$

In the case of (4.2.7)-(4.2.8), the following function spaces are involved

$$\begin{array}{ccccc} & \mathbf{curl} & & \mathbf{div} & \\ H^1(\Omega) & \longrightarrow & H(\text{div}, \Omega) & \longrightarrow & L^2(\Omega) \\ \cup & & \cup & & \cup \\ V & \longrightarrow & W_{\text{div}} & \longrightarrow & X \end{array} \quad (4.3.10)$$

In order to keep the specific features of Maxwell's equations at the discrete level we need to construct finite dimensional subspaces endowed with the same structure. The involved discrete spaces are denoted by $X \subset H^1(\Omega)$, $W_{\text{curl}} \subset H(\text{rot}, \Omega)$, $W_{\text{div}} \subset H(\text{div}, \Omega)$ and $V \subset L^2(\Omega)$. When discretizing the first variational formulation (4.2.5)-(4.2.6), we shall look for $(\mathbf{E}_h, H_h) \in W_{\text{curl}} \times V$ and when discretizing the second variational formulation (4.2.7)-(4.2.8), we shall look for $(\mathbf{E}_h, H_h) \in W_{\text{div}} \times X$.

4.3. Construction of the finite element spaces

4.3.1 Spline finite elements on patch grids

We shall now start constructing the actual subspaces $X \subset H^1(\Omega)$, $W_{curl} \subset H(\text{curl}, \Omega)$, $W_{div} \subset H(\text{div}, \Omega)$ and $V \subset L^2(\Omega)$.

Our discrete space will be constructed using B-splines.

The key point of our method is the use of the recursion formula for the derivatives:

$$N_i^{p'}(t) = p \left(\frac{N_i^{p-1}(t)}{t_{i+p} - t_i} - \frac{N_{i+1}^{p-1}(t)}{t_{i+p+1} - t_{i+1}} \right). \quad (4.3.11)$$

It will be convenient to introduce the notation $D_i^p = p \frac{N_i^{p-1}(t)}{t_{i+p} - t_i}$. Then the recursion formula for derivative simply becomes

$$N_i^{p'}(t) = D_i^p(t) - D_{i+1}^p(t). \quad (4.3.12)$$

Discrete vector fields on a rectangular domain.

Let us first consider a rectangular domain Ω . We consider the following discrete functional spaces

$$V = \text{span}\{N_i^p(x)N_j^p(y), 1 \leq i \leq N_x, 1 \leq j \leq N_y\},$$

$$W_{div} = \text{span} \left\{ \begin{pmatrix} N_i^p(x)D_j^p(y) \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ D_i^p(x)N_j^p(y) \end{pmatrix}, 1 \leq i \leq N_x, 1 \leq j \leq N_y \right\},$$

$$W_{curl} = \text{span} \left\{ \begin{pmatrix} D_i^p(x)N_j^p(y) \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ N_i^p(x)D_j^p(y) \end{pmatrix}, 1 \leq i \leq N_x, 1 \leq j \leq N_y \right\},$$

$$X = \text{span}\{D_i^p(x)D_j^p(y), 1 \leq i \leq N_x, 1 \leq j \leq N_y\}.$$

As proved in the article by Buffa et al, [16, 18, 17] these spaces verify the same exact property as the spaces they approximate.

Vector field transformations.

Let us now define coordinate changes conserving either the curl or the divergence of a vector field.

Let us start from a vector field $\underline{\Psi}(\xi, \eta) = (\underline{\Psi}^{(1)}(\xi, \eta), \underline{\Psi}^{(2)}(\xi, \eta))^T$ defined on the parametric domain \tilde{Q} .

Using the transformation formula (B.0.3) for the vector fields of W_{div} which conserves the divergence, using the diffeomorphism $G = F^{-1}$, we get $\Psi^{(1)} = \frac{1}{\Delta}(\alpha_2 \Psi^{(2)} + \alpha_1 \Psi^{(1)})$ and $\Psi^{(2)} = \frac{1}{\Delta}(\beta_2 \Psi^{(2)} + \beta_1 \Psi^{(1)})$.

So let $\Psi = (\Psi^{(1)}, \Psi^{(2)})^T$ be a function in W_{div} , and $\underline{\Psi} = (\underline{\Psi}^{(1)}, \underline{\Psi}^{(2)})^T$ be a function in \widetilde{W}_{div} .

To save the divergence property, the corresponding space of \widetilde{W}_{div} on the physical domain is

$$W_{div} = \{ \Psi := (\frac{1}{\Delta}(\alpha_1 \Psi^{(1)} + \alpha_2 \Psi^{(2)}), \frac{1}{\Delta}(\beta_1 \Psi^{(1)} + \beta_2 \Psi^{(2)}))^T, \quad \underline{\Psi} \in \widetilde{W}_{div} \}$$

therefore,

$$W_{div} = \text{span} \left\{ \frac{1}{\Delta} \tilde{N}_i^p(\xi) \tilde{D}_j^p(\eta) \begin{pmatrix} \alpha_1 \\ \beta_1 \end{pmatrix}, \frac{1}{\Delta} \tilde{D}_i^p(\xi) \tilde{N}_j^p(\eta) \begin{pmatrix} \alpha_2 \\ \beta_2 \end{pmatrix} \right\}.$$

4.3. Construction of the finite element spaces

4.3.2 The Discrete Equations

Let us now express the equation satisfied by the approximations \mathbf{E}_h, H_h when using each of the variational formulations we introduced. Let's start with (4.2.7)-(4.2.8). In this case we look for $\mathbf{E}_h \in W_{div}$ and $H_h \in H^1(\Omega)$. We first notice that due to the exact sequence property we have $div \mathbf{E}_h \in X$.

Let us denote by

$$\boldsymbol{\psi}_{i,j}^1 = \begin{pmatrix} N_i^p(x) D_j^{p-1}(y) \\ 0 \end{pmatrix}, \quad \boldsymbol{\psi}_{i,j}^2 = \begin{pmatrix} 0 \\ D_i^{p-1}(x) N_j^p(y) \end{pmatrix}$$

we have

$$W_{div} = \text{span} \{ \boldsymbol{\psi}_{i,j}^1, \boldsymbol{\psi}_{i,j}^2, \quad 1 \leq i \leq N_x, 1 \leq j \leq N_y \},$$

Denoting the components of \mathbf{E}_h by (E_h^x, E_h^y) , we have

$$E_h^x(t, x, y) = \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} e_{i,j}^x(t) N_i^p(x) D_j^p(y), \quad E_h^y(t, x, y) = \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} e_{i,j}^y(t) D_i^p(x) N_j^p(y),$$

and

$$H_h^z(t, x, y) = \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} h_{i,j}^z(t) N_i^p(x) N_j^p(y).$$

Using these expansions on the finite element bases, we can compute explicitly, for

$$\text{rot } H_h = \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} h_{i,j}^z(t) \begin{pmatrix} N_i^p(x) N_j^{p'}(y) \\ -N_i^{p'}(x) N_j^p(y) \end{pmatrix}$$

we now use the formula (4.3.11),

$$\begin{aligned} \text{rot } H_h &= \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} h_{i,j}^z(t) \begin{pmatrix} N_i^p(x) (D_j^{p-1}(y) - D_{j+1}^{p-1}(y)) \\ -(D_i^{p-1}(x) - D_{i+1}^{p-1}(x)) N_j^p(y) \end{pmatrix} \\ &= \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} h_{i,j}^z(t) \{ \boldsymbol{\psi}_{i,j}^1 - \boldsymbol{\psi}_{i,j+1}^1 - \boldsymbol{\psi}_{i,j}^2 + \boldsymbol{\psi}_{i+1,j}^2 \}. \end{aligned}$$

We will use the following convention: for a knot vector $T = (t_i)_{1 \leq i \leq N+k}$ generating the "N" B-splines of order k , $\{N_i^k, 1 \leq i \leq N\}$, we put $N_j^k := 0$ for any $j > N$.

By making a change of index in the sum, we get

$$\text{rot } H_h = \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} h_{i,j}^z \{ \boldsymbol{\psi}_{i,j}^1 - \boldsymbol{\psi}_{i,j}^2 \} - \sum_{i=1}^{N_x} \sum_{j=2}^{N_y+1} h_{i,j-1}^z \boldsymbol{\psi}_{i,j}^1 + \sum_{i=2}^{N_x+1} \sum_{j=1}^{N_y} h_{i-1,j}^z \boldsymbol{\psi}_{i,j}^2$$

therefore,

$$\begin{aligned} \text{rot } H_h &= \sum_{i=2}^{N_x} \sum_{j=2}^{N_y} (h_{i,j}^z - h_{i,j-1}^z) \boldsymbol{\psi}_{i,j}^1 - (h_{i,j}^z - h_{i-1,j}^z) \boldsymbol{\psi}_{i,j}^2 \\ &+ \sum_{j=2}^{N_y} \{ h_{1,j}^z - h_{1,j-1}^z \} \boldsymbol{\psi}_{1,j}^1 + \sum_{i=2}^{N_x} \{ h_{i-1,1}^z - h_{i,1}^z \} \boldsymbol{\psi}_{i,1}^2 + h_{1,1}^z \boldsymbol{\psi}_{1,1}^1 - h_{1,1}^z \boldsymbol{\psi}_{1,1}^2. \end{aligned}$$

4.4. Leap Frog scheme's stability

In case of periodic boundary conditions, the discrete Ampere's law (4.2.7) can be written

$$-\dot{\mathbf{e}} + R\mathbf{h}^z = M_W^{-1}\mathbf{j}, \quad (4.3.13)$$

where \mathbf{h}^z , e^x , e^y the vectors of spline coefficients, M_W is the mass matrix for W_{div} and R consists of two blocks of the discrete derivatives in the x and y directions of these vectors.

Remark We've shown that we can write $K^T = M_W R$, where K is the rotational matrix involved in the classical formulation (without current density field):

$$\begin{cases} M_W \dot{\mathbf{e}} = K\mathbf{h} \\ M_V \dot{\mathbf{h}} = -K^T \mathbf{e} \end{cases}$$

therefore, the linear system writes:

$$\begin{cases} \dot{\mathbf{e}} = R\mathbf{h} \\ M_V \dot{\mathbf{h}} = -K^T \mathbf{e} \end{cases} \quad (4.3.14)$$

A discrete Faraday's law can be obtained from the variational formulation (4.2.6) which can be written

$$\dot{\mathbf{h}}^z + d_x e^y - d_y e^x = 0, \quad (4.3.15)$$

where d_x and d_y the discrete derivatives in the x and y directions.

We have thus obtained matrix differential equations on the vectors of spline coefficients that can be solved by any appropriate ODE solver. For example, for a second order time discretization, a leap-frog or Verlet scheme is well adapted.

4.4 Leap Frog scheme's stability

In this section we will give a preview of the time discretization scheme we used to solve the linear system obtained and some classical results. Let us first, just recall the general formulation of the LF scheme. For the linear system

$$\begin{cases} M_W \dot{\mathbf{e}} = K\mathbf{h} \\ M_V \dot{\mathbf{h}} = -K^T \mathbf{e} \end{cases}$$

the LF time discretization of order N is given by

$$\begin{cases} M_W \frac{E^{n+1} - E^n}{\Delta t} = K_N H^{n+\frac{1}{2}} \\ M_V \frac{H^{n+\frac{3}{2}} - H^{n+\frac{1}{2}}}{\Delta t} = -K_N^T E^{n+1} \end{cases}$$

where

$$\begin{cases} K_N = K & , N = 2 \\ K_N = K \left(I - \frac{\Delta t^2}{24} M_W^{-1} K M_V^{-1} K^T \right) & , N = 4 \end{cases}$$

We define the global energy by $\mathcal{E}^n := \frac{1}{2} \{ (E^n)^T M_W E^n + (H^{n-\frac{1}{2}})^T M_V H^{n+\frac{1}{2}} \}$ We have the following lemma:

Lemma 4.4.1 *The global energy is stationary, i.e $\mathcal{E}^{n+1} = \mathcal{E}^n$*

4.5. Numerical results

Proof Let us put $E^{n+\frac{1}{2}} = \frac{E^{n+1}+E^n}{2}$. Then we have,

$$\begin{aligned}\mathcal{E}^{n+1} - \mathcal{E}^n &= \frac{1}{2}((E^{n+1})^T M_W E^{n+1} + (H^{n+\frac{1}{2}})^T M_V H^{n+\frac{3}{2}} - \frac{1}{2}((E^n)^T M_W E^n - (H^{n-\frac{1}{2}})^T M_V H^{n+\frac{1}{2}})) \\ &= (E^{n+1})^T M_W E^{n+\frac{1}{2}} - (E^n)^T M_W E^{n+\frac{1}{2}} + \frac{1}{2}(H^{n-\frac{1}{2}})^T M_V (H^{n+\frac{3}{2}} - H^{n+\frac{1}{2}}) \\ &= (E^{n+\frac{1}{2}})^T M_W (E^{n+1} - E^n) + \frac{1}{2}(H^{n-\frac{1}{2}})^T M_V (H^{n+\frac{3}{2}} - H^{n+\frac{1}{2}})\end{aligned}$$

using the time discretization scheme, we have:

$$\mathcal{E}^{n+1} - \mathcal{E}^n = \Delta t (E^{n+\frac{1}{2}})^T K_N H^{n+\frac{1}{2}} - \Delta t (H^{n+\frac{1}{2}})^T K_N E^{n+\frac{1}{2}}$$

Therefore, we have $\mathcal{E}^{n+1} - \mathcal{E}^n = 0$

The stability of the scheme depends on the global energy, which must be a positive quadratic form to ensure stability.

Lemma 4.4.2 \mathcal{E}^n is a positive quadratic form if $\Delta t \leq \frac{2}{d_N}$, where $d_N = \|M_V^{-\frac{1}{2}} K_N^T M_W^{-\frac{1}{2}}\|$

Proof

$$\begin{aligned}\mathcal{E}^n &= \frac{1}{2}(E^n)^T M_W E^n + \frac{1}{2}(H^{n-\frac{1}{2}})^T M_V H^{n+\frac{1}{2}} \\ &= \frac{1}{2}(E^n)^T M_W E^n + (H^{n-\frac{1}{2}})^T M_V H^{n-\frac{1}{2}} - \frac{\Delta t}{2}(H^{n-\frac{1}{2}})^T M_V E^n \\ &\geq \frac{1}{2}\|M_W^{\frac{1}{2}}\|^2 + \frac{1}{2}\|M_V^{\frac{1}{2}}\|^2 - \frac{\Delta}{2} |(H^{n-\frac{1}{2}})^T M_V^{\frac{1}{2}} M_V^{-\frac{1}{2}T} K_N M_W^{-\frac{1}{2}} M_W^{\frac{1}{2}} E^n| \\ &\geq \frac{1}{2}\|M_W^{\frac{1}{2}}\|^2 + \frac{1}{2}\|M_V^{\frac{1}{2}}\|^2 - \frac{\Delta d_N}{2} \{\|M_V^{\frac{1}{2}} H^{n-\frac{1}{2}}\|^2 + \|M_W^{\frac{1}{2}} E^n\|^2\} \\ &\geq \frac{1}{2}\|M_W^{\frac{1}{2}}\|^2 + \frac{1}{2}\|M_V^{\frac{1}{2}}\|^2 - \frac{\Delta d_N}{4} \|M_V^{\frac{1}{2}} H^{n-\frac{1}{2}}\| \|M_W^{\frac{1}{2}} E^n\| \\ &\geq \frac{1}{2} (1 - \frac{\Delta t d_N}{2}) \{\|M_W^{\frac{1}{2}}\|^2 + \|M_V^{\frac{1}{2}}\|^2\}\end{aligned}$$

The last quantity is positive if $1 \geq \frac{\Delta t d_N}{2}$.

Finally, we can take $CFL_N^{th} = \frac{2}{hd_N}$. As proved in [40], we have $CFL_4^{th} \simeq 2.85 CFL_2^{th}$.

4.5 Numerical results

In [16, 18, 17], the authors construct adequate interpolators that achieve the optimal rates (please see the section 5 of [17]). Thanks to the theory developed in literature (see e.g. [90]), we can expect a convergence order of p for the electric field, and $p + 1$ for the magnetic field with our discrete spaces defined in section 4.3.2.

In all tests, we insert knots with a multiplicity of 1. To solve the linear equation $M_V \dot{h} = -K^T e$ involved in (4.3.14), we used the band solver from LAPACK, with the routines DGBTRS, DGBTRF. We have also tested PASTIX, and it reduces considerably the computational time.

4.5. Numerical results

4.5.1 Test case 1: square

The analytical solution in this case is:

$$H^z = \cos(k_1 x + \phi_1) \cos(k_2 y + \phi_2) \cos(\omega t), \quad (4.5.16)$$

$$E^x = -\frac{k_2}{\omega} \cos(k_1 x + \phi_1) \sin(k_2 y + \phi_2) \sin(\omega t), \quad (4.5.17)$$

$$E^y = \frac{k_2}{\omega} \sin(k_1 x + \phi_1) \cos(k_2 y + \phi_2) \sin(\omega t). \quad (4.5.18)$$

For our test we took a computational domain of size $[0, 2\pi] \times [0, 2\pi]$ and

$$k_1 = k_2 = 1, \quad \phi_1 = \phi_2 = 0, \quad \omega = \pi$$

This test enables to check the L^2 norm of the error (in log scale) between the numerical and analytical solution, with respect to the parameter $h = \max\{\text{diam}(\tilde{Q})\}$, for different orders of the basis functions (from order 3 to order 6). This is shown in Figure 4.1. We verify that the slopes of the different curves correspond to the order of the basis functions. In particular, for high orders, the machine precision is achieved. For validation, we solved the Maxwell's equation for $N = 16, 32, 64, 128$, while keeping $\frac{\Delta t}{h}$ constant, and computed the error at the time $N_{iter}\Delta t = C$, with N_{iter} is the number of iterations, taken to keep the same final time.

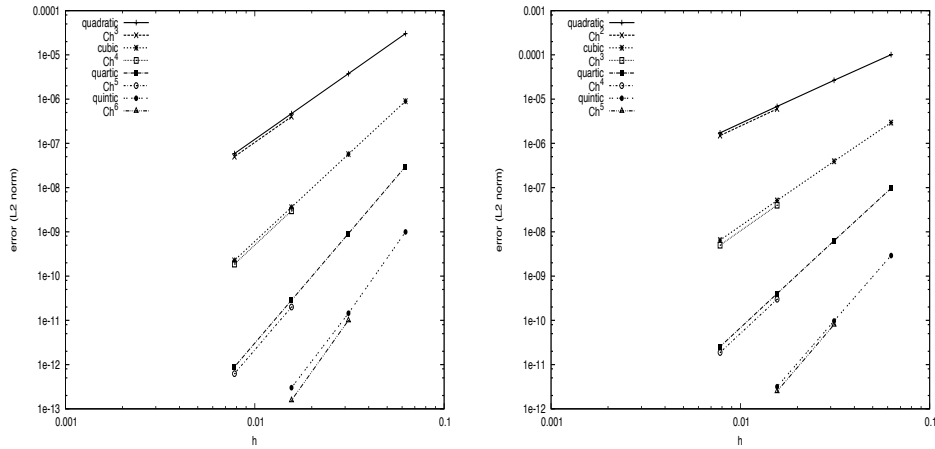


Figure 4.1: Square test: the L^2 norm error for (left) the magnetic field, (right) the electric field

The k-refinement strategy helped as to reduce the number of degrees of freedom as we can see it in Figure 4.2, where the number of degrees of freedom was reduced by a factor of 6 between multiplicity 2 (corresponding to quadratic Lagrange finite elements) and 1 keeping the same accuracy. The price to pay, is that we increased the support of the basis functions compared to the classical finite element method. But as we said before, this is the best we can do using splines functions, the B-splines have the minimal support that we can get if we try to increase the regularity of our basis.

Tables 4.1 and 4.2 show the CFL numbers for different B-splines degree $p = 2, \dots, 5$. We see that the CFL decreases with the B-splines degree and the knots multiplicity.

4.5. Numerical results

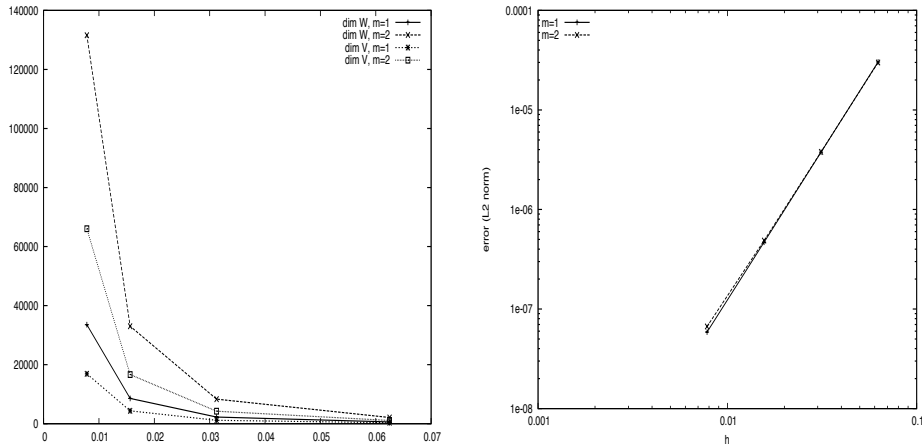


Figure 4.2: Square test: (left) the dimension of the discrete spaces W_h and V_h , (right) the L^2 norm error for the electric field, where the vector knots are multiplicity $m = 1, 2$ for quadratic B-splines

	$LF2^{Th}$	$LF2^{num}$	$LF4^{Th}$	$LF4^{num}$
$p = 2$	0.3044	0.3056	0.8676	0.8720
$p = 3$	0.2058	0.1840	0.5866	0.5872
$p = 4$	0.1496	0.1520	0.4265	0.4272
$p = 5$	0.1151	0.1168	0.3281	0.3280

Table 4.1: Test case 1: CFL numbers (theoretical and numerical values), for splines of degree $p = 2, \dots, 5$

	$m=1$	$m=2$
$p = 2$	0.8720	0.5200
$p = 3$	0.5872	0.4224
$p = 4$	0.4272	0.3056
$p = 5$	0.3280	0.2304

Table 4.2: Test case 1: CFL, using LF4, for splines of degree $p = 2, \dots, 5$ for singular knots ($m = 1$), and doubled knots ($m = 2$)

	$N = 16$	$N = 32$	$N = 64$
$p = 2$	11.35	48.67	239.25
$p = 3$	24.74	105.10	471.10
$p = 4$	50.72	211.09	919.90
$p = 5$	91.61	392.61	1708.61

Table 4.3: Test case 1: computation time in seconds after 10000 iterations. Computations were done on MacBook 2GHz Intel Core 2 Duo, Memory 2 Go 667 MHz DDR2, using the band solver from LAPACK

$LF2^{num}$ and $LF4^{num}$ are obtained numerically, by fixing h and increasing Δt until divergence (let us denote it Δt_{max}). The CFL number is therefore the value $\frac{\Delta t_{max}}{h}$. This was done for different values of h .

It is well known [5] that for rectangular meshes, there is a condition to get the optimal

4.5. Numerical results

convergence rate. Otherwise, we will see a degradation of this rate. In our case, as we are only using a cartesian meshes for the parametric domain (patch), we can recover the same property by changing the regularity of the mapping F . In Figure 4.3, we construct a mapping F by elevating the B-spline degree, and repositioning 4 control points. We can see, that this operation has generated a discontinuity of the jacobian of F . This surely will cause a degradation of the convergence rate, as it is shown in Figure 4.3. An important question, could be how can we weaken the regularity constraint on the mapping?

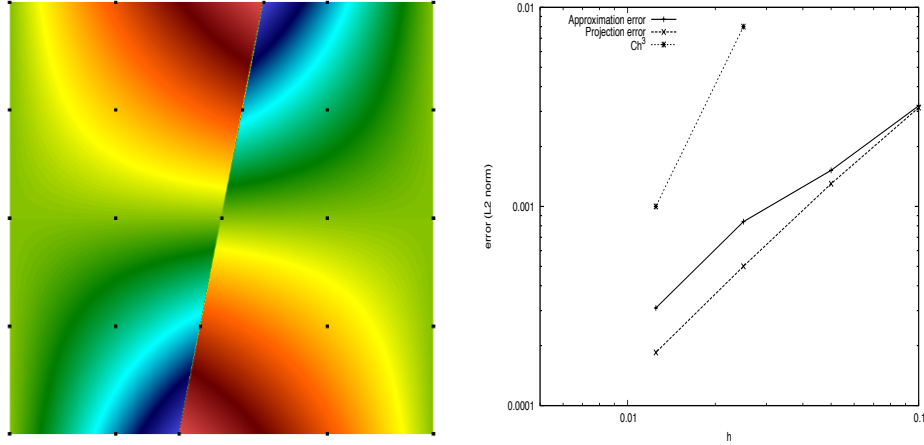


Figure 4.3: (left) The jacobian of the mapping F and control points. (right) degradation of the convergence rate

4.5.2 Test case 2: circular wave guide

The analytical solution, in the polar coordinates, in this case is:

$$\begin{aligned} H^z(r, \theta) &= -\cos(\omega t + \theta) \{J_1(\omega r) + aY_1(\omega r)\} \\ E^\theta(r, \theta) &= \frac{1}{2} \sin(\omega t + \theta) \{J_0(\omega r) - J_2(\omega r) + a(Y_0(\omega r) - Y_2(\omega r))\} \\ E^r(r, \theta) &= -\frac{1}{\omega r} \cos(\omega t + \theta) \{J_1(\omega r) + aY_1(\omega r)\} \end{aligned}$$

where J_n, Y_n are the first and the second Bessel functions of order n .

For our test we took

$$r_{min} = 0.65138750344695903414, \quad r_{max} = 0.99000418530735846839, \quad a = 1.0, \quad \omega = 3\pi$$

In the sequel, we present two different ways to solve this problem.

Polar mapping, with periodic splines

Under a polar mapping, we use uniform periodic B-splines in the theta direction and uniform B-splines with open knots in the radial direction. The use of uniform B-splines allows to reduce the number of degrees of freedom. Therefore, we can achieve an error of 10^{-11} , using quintic B-splines, with only 64×64 meshes, and $\dim W_h = 8640$, $\dim V_h =$

4.5. Numerical results

4352. A precision of 10^{-7} is achieved, using quintic B-splines, with only 16×16 meshes. In figures 4.4 and 4.5, we have plotted the solution and the convergence order, using a *LF4* time scheme.

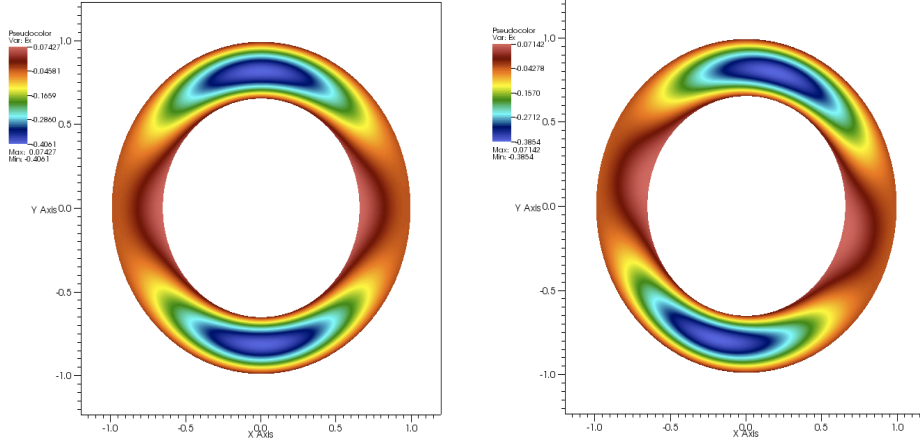


Figure 4.4: Circular wave guide test: (left) at $t = 0$, (right) after 20 iterations, for $N = 64$, $p = 3$

Polar mapping, with C^0 -periodic splines

Here, we use a polar mapping, which is only C^0 . In fact, rather than taking uniform *B-splines* in the θ -direction, we impose strongly the periodicity, assuming that control points must coincide on the extremities of each (closed) curve in the θ -direction. We shall observe, a degradation of the convergence order. In figures 4.6, 4.7 and 4.8, we have plotted the L^2 error norm and computed the convergence order, at $t = 0.0, 0.05$ and 0.5 . This shows the degradation of the accuracy, depending on the spline degree. Notice that the C^0 -periodicity does not have any impact at $t = 0.0$, *i.e* the projection over the discrete spaces. We also observe that the accuracy degradation is more important using *B-splines* of odd degree (the reference is for the magnetic field).

Remark 4.5.1 *In order to use a C^0 condition, we must have only Bernstein polynomials.*

4.5.3 Test case 3: Silver-Muller condition

In the case of the Silver-Muller condition the discrete Faraday's equation of (4.2.8) is written

$$\partial_t M_V h^z + K e + \Gamma h^z = 0$$

where Γ is the mass matrix of the discrete V_h on the boundary that implements Silver-Muller condition *i.e* $\int_{\partial\Omega_{SM}} N_{i,j} N_{i',j'}$

In this test we see the evolution of an electromagnetic wave (figure 4.9), under Silver-Muller condition on both the internal and external boundary. At $t = 0$, we took $E_x = u(x)u(y)'$ and $E_Y = u(x)'u(y)$, with $u(x) = \exp(-\frac{(x-m)^2}{2\sigma^2})$.

4.6. H-rot formulation

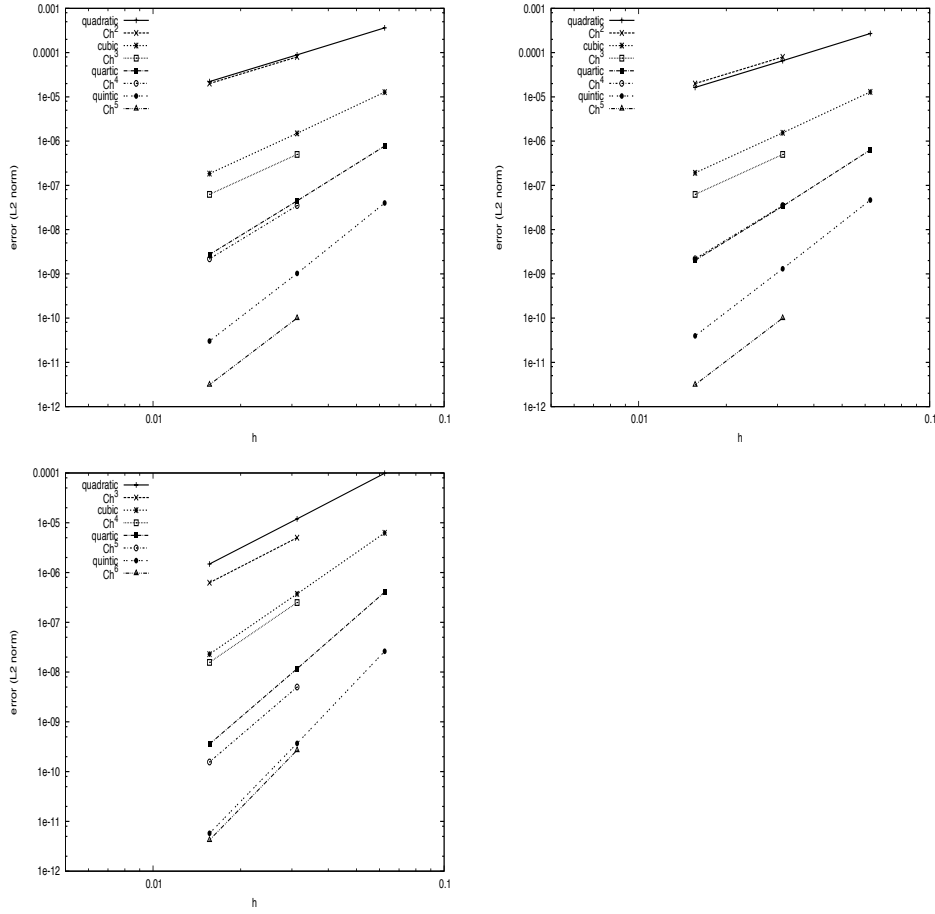


Figure 4.5: Circular wave guide test: the L^2 norm error for, first line E^x (left) and E^y (right) components of the electric field, second line the magnetic field

4.6 H-rot formulation

In this section, we will use the diagram 4.3.9 to construct a discrete DeRham sequence. As noticed before, the spaces in the patch, involved in this case are:

$$\begin{aligned}
 V &= \text{span}\{N_i^p(x)N_j^p(y), 1 \leq i \leq N_x, 1 \leq j \leq N_y\}, \\
 W_{curl} &= \text{span}\left\{\begin{pmatrix} D_i^p(x)N_j^p(y) \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ N_i^p(x)D_j^p(y) \end{pmatrix}, 1 \leq i \leq N_x, 1 \leq j \leq N_y\right\}, \\
 X &= \text{span}\{D_i^p(x)D_j^p(y), 1 \leq i \leq N_x, 1 \leq j \leq N_y\}.
 \end{aligned}$$

As in the case of the **H-div** formulation, we will need to transform carefully the vector functions in order to conserve their properties. In this case,

$$W_{curl} = \text{span}\left\{\frac{1}{\Delta}\tilde{D}_i^p(\xi)\tilde{N}_j^p(\eta)\begin{pmatrix} \beta_2 \\ -\beta_1 \end{pmatrix}, \frac{1}{\Delta}\tilde{N}_i^p(\xi)\tilde{D}_j^p(\eta)\begin{pmatrix} -\alpha_2 \\ \alpha_1 \end{pmatrix}\right\}. \quad (4.6.19)$$

After discretization, we get the linear system :

$$\begin{cases} M^W \dot{\mathbf{e}} = K \mathbf{h} \\ M^V \dot{\mathbf{h}} = -K^T \mathbf{e} \end{cases} \quad (4.6.20)$$

4.7. Axisymmetric Variational formulation of the 2D Maxwell's equation

where,

$$\begin{aligned} M_{b,b'}^W &= \int_{\Omega} \Psi_b \cdot \Psi_{b'} d\Omega, \\ M_{b,b'}^V &= \int_{\Omega} \varphi_b \varphi_{b'} d\Omega, \\ K_{b,b'} &= \int_{\Omega} \varphi_b \text{rot} \Psi_{b'} d\Omega. \end{aligned}$$

with the usual notation:

$$\begin{aligned} W_{curl} &= \text{span} \{ \Psi_b \}. \\ V &= \text{span} \{ \varphi_b \}. \end{aligned}$$

Numerical results

Let's go back to the test presented in section 4.5.1, with the analytical solutions given by equations 4.5.16, 4.5.17 and 4.5.18. Results are given in figure 4.10.

4.7 Axisymmetric Variational formulation of the 2D Maxwell's equation

In this section, we shall investigate different formulation to solve the axisymmetric case. More details about the axisymmetric Maxwell equations can be found in [6, 7].

Maxwell's equations

The 2D Maxwell equations, in the TE mode, are

$$\left\{ \begin{array}{l} -\frac{\partial \mathbf{E}}{\partial t} + \text{rot} B = \mathbf{J}, \\ \frac{\partial B}{\partial t} + \text{rot} \mathbf{E} = 0, \\ \text{div} \mathbf{E} = \rho. \end{array} \right.$$

For the axisymmetric geometry, we have:

$$\mathbf{E} = \begin{pmatrix} E_z \\ E_r \end{pmatrix}, B = B_\theta, \text{rot} \mathbf{E} = \partial_z E_r - \partial_r E_z, \text{div} \mathbf{E} = \frac{1}{r} \partial_r (r E_r) + \partial_z E_z \text{ et } \text{rot} B = \begin{pmatrix} \frac{1}{r} \partial_r (r B_\theta) \\ -\partial_z B_\theta \end{pmatrix}.$$

Operators expressions

We have,

$$\begin{aligned} \mathbf{E} \cdot \Psi &= E_z \Psi_z + E_r \Psi_r, \\ \mathbf{J} \cdot \Psi &= J_z \Psi_z + J_r \Psi_r, \\ \mathbf{E} \cdot \text{rot} \phi &= E_z \frac{1}{r} \partial_r (r \phi) - E_r \partial_z \phi, \\ \text{rot} B \cdot \Psi &= \Psi_z \frac{1}{r} \partial_r (r B) - \Psi_r \partial_z B, \end{aligned}$$

4.7. Axisymmetric Variational formulation of the 2D Maxwell's equation

Validation test

For numerical validation, we solve the Maxwell's equations on a rectangle $z \in (0, L)$ and $r \in (0, R)$. Let m be the mode, the analytic solution is

$$\begin{aligned} H(z, r) &= \frac{\omega \lambda_p}{c R} \sqrt{\frac{\epsilon_0}{\mu_0}} J_1\left(\frac{\lambda_p r}{R}\right) \cos\left(\frac{m\pi z}{L}\right) \cos(\omega t), \\ E^z(z, r) &= \frac{1}{\mu_0} \left(\frac{\lambda_p}{R}\right)^2 J_0\left(\frac{\lambda_p r}{R}\right) \cos\left(\frac{m\pi z}{L}\right) \sin(\omega t), \\ E^r(z, r) &= \frac{1}{\mu_0} \frac{m\pi \lambda_p}{L R} J_1\left(\frac{\lambda_p r}{R}\right) \sin\left(\frac{m\pi z}{L}\right) \sin(\omega t), \end{aligned}$$

where J_n are Bessel functions of the first kind order n . λ_p is the p^{th} zero of J_0 . In practice, we will take $L = R = 1$.

In the sequel, we present different variational formulations.

The expected convergence order is $p + 1$ for the magnetic field and p for the electric field.

4.7.1 Discrete equations - 1st formulation

As known, to derive a variational formulation, we will relax one equation (Ampère or Faraday), and keep the other one in the strong form. Multiplying the two equations by a test function and integrating on the physical domain Ω , we obtain: $\forall \Psi \in H(\text{div}, \Omega)$,

$$\begin{aligned} \int_{\Omega} -\frac{\partial \mathbf{E}}{\partial t} \cdot \Psi \, dX + \int_{\Omega} \text{rot } B \cdot \Psi \, dX &= \int_{\Omega} \mathbf{J} \cdot \Psi \, dX, \\ -\frac{\partial}{\partial t} \int_{\Omega} \mathbf{E} \cdot \Psi \, dX + \int_{\Omega} \text{rot } B \cdot \Psi \, dX &= \int_{\Omega} \mathbf{J} \cdot \Psi \, dX. \end{aligned}$$

We have, for the Faraday equation, $\forall \phi \in H^1(\Omega)$,

$$\int_{\Omega} \frac{\partial B}{\partial t} \phi \, dX + \int_{\Omega} \text{rot } \mathbf{E} \phi \, dX = 0.$$

Now by relaxing this equation, and if we are dealing with perfectly conducting boundary conditions, we have

$$\frac{\partial}{\partial t} \int_{\Omega} B \phi \, dX + \int_{\Omega} \mathbf{E} \cdot \text{rot } \phi \, dX = 0,$$

where we used the Green formula:

$$\int_{\Omega} (\text{rot } G) \cdot \mathbf{F} \, dX = \int_{\Omega} G \text{rot } \mathbf{F} \, dX - \int_{\Gamma} (G \times \mathbf{n}) \cdot \mathbf{F} \, dS, \quad \forall \mathbf{F} \in H(\text{curl}, \Omega), \quad \forall G \in H^1(\Omega).$$

In the case of axisymmetric geometry, the measure dX considered is simply $dX = r \, dr \, dz$.

Therefore,

$$-\frac{\partial}{\partial t} \int_{\Omega} (E_z \Psi_z + E_r \Psi_r) \, r \, dr \, dz + \int_{\Omega} (\Psi_z \partial_r (rB) - r \Psi_r \partial_z B) \, dr \, dz = \int_{\Omega} (J_z \Psi_z + J_r \Psi_r) \, r \, dr \, dz,$$

4.7. Axisymmetric Variational formulation of the 2D Maxwell's equation

and,

$$\frac{\partial}{\partial t} \int_{\Omega} B \phi r dr dz + \int_{\Omega} (E_z \partial_r (r\phi) - r E_r \partial_z \phi) dr dz = 0.$$

Let $W_{div} = \text{span}\{\varphi_b, b \in \Lambda_E\}$ the basis for W_{div} , the discrete space associated to the electrical field. Let $V = \text{span}\{\Psi_{b'}, b' \in \Lambda_H\}$ the basis of V , it discretizes the space associated to the magnetic field.

Mass matrix for the electrical field

$$M_{b,b'}^W = \int_{\Omega} \Psi_b \cdot \Psi_{b'} r dr dz.$$

Mass matrix for the magnetic field

$$M_{b,b'}^V = \int_{\Omega} \varphi_b \varphi_{b'} r dr dz.$$

Matrix for the curl

$$\begin{aligned} K_{b,b'} &= \int_{\Omega} \text{rot} \varphi_b \cdot \Psi_{b'} r dr dz = \int_{\Omega} (r \partial_r \varphi_b + \varphi_b) \psi_{b'}^z - (r \partial_z \varphi_b) \psi_{b'}^r dr dz \\ &= \int_{\Omega} r (\partial_r \varphi_b \psi_{b'}^z - \partial_z \varphi_b \psi_{b'}^r) dr dz + \int_{\Omega} \varphi_b \psi_{b'}^z dr dz. \end{aligned}$$

This is the contribution of two terms. The first one " $r(\partial_r \varphi_b \psi_{b'}^z - \partial_z \varphi_b \psi_{b'}^r)$ " which differs from the cartesian formulation by a multiplication by r . The second one " $\varphi_b \psi_{b'}^z$ " is due to the axisymmetric geometry.

Therefore, we obtain the classical system:

$$M^W \partial_t [E] = K [B], \quad (4.7.21)$$

$$M^V \partial_t [B] = -K^T [E] \quad (4.7.22)$$

In figures 4.11 and 4.12, we have plotted the L^2 error norm and computed the convergence order, at $t = 0.0$ and 1.0 . Observing the convergence orders, we may suppose that :

- the convergence order for B is :

$$\begin{cases} p - \frac{1}{2}, & \text{if } p \text{ is even} \\ p + \frac{1}{2}, & \text{if } p \text{ is odd} \end{cases}$$

- the convergence order for E_z is $p + 1$,
- the convergence order for E_r is the minimum of the two previous ones.

4.7. Axisymmetric Variational formulation of the 2D Maxwell's equation

4.7.2 Discrete equations - 2nd formulation

In this section, we are interested in solving the Maxwell's equations on a square domain. We shall consider a scalar version for the variational formulation, based on one discrete space \mathcal{V}_h spanned by *B-splines* (this can also be done using *NURBS*). Let's go back to Maxwell's equations without source terms:

$$\begin{cases} -\frac{\partial \mathbf{E}}{\partial t} + \mathbf{rot} B = 0, \\ \frac{\partial B}{\partial t} + \mathbf{rot} \mathbf{E} = 0. \end{cases}$$

Discrete equations

Let $\varphi \in \mathcal{V}_h$, we have :

$$\partial_t \int B \varphi + \int \mathbf{rot} \mathbf{E} \varphi = 0$$

using the Green's formulae we get:

$$\partial_t \int B \varphi + \int \mathbf{E} \cdot \mathbf{rot} \varphi = 0$$

now let us compute the term $\int \mathbf{E} \cdot \mathbf{rot} \varphi$. We have,

$$\begin{aligned} \int \mathbf{E} \cdot \mathbf{rot} \varphi &= \int E_z \left(\frac{1}{r} \varphi + \partial_r \varphi \right) - E_r \partial_z \varphi r dr dz \\ &= \int E_z \partial_r \varphi r dr dz - \int E_r \partial_z \varphi r dr dz + \int E_z \varphi dr dz \end{aligned}$$

on the other hand, we have for $\varphi_{b_1}, \varphi_{b_2} \in \mathcal{V}_h$:

$$\begin{aligned} \int \mathbf{rot} B \cdot \begin{pmatrix} \varphi_{b_1} \\ \varphi_{b_2} \end{pmatrix} &= \int \varphi_{b_1} \left(\frac{1}{r} B + \partial_r B \right) - \varphi_{b_2} \partial_z B r dr dz \\ &= \int \varphi_{b_1} \partial_r B r dr dz - \int \varphi_{b_2} \partial_z B r dr dz + \int \varphi_{b_1} B dr dz \end{aligned}$$

now let us expand E_r, E_z and B on \mathcal{V}_h , we have:

$$\int E_z \partial_r \varphi_b r dr dz = \sum_{b'} [E_z]^{b'} \int \varphi_{b'} \partial_r \varphi_b r dr dz = (D_r[E_z])_b \quad (4.7.23)$$

and,

$$\int E_r \partial_z \varphi_b r dr dz = \sum_{b'} [E_r]^{b'} \int \varphi_{b'} \partial_z \varphi_b r dr dz = (D_z[E_r])_b \quad (4.7.24)$$

and,

where we define the matrices,

$$(D_z)_{b,b'} = \int \varphi_b \partial_z \varphi_{b'} r dr dz, \quad \text{and} \quad (D_r)_{b,b'} = \int \varphi_b \partial_r \varphi_{b'} r dr dz \quad (4.7.25)$$

$$(M)_{b,b'} = \int \varphi_{b'} \varphi_b dr dz, \quad \text{and} \quad (M_r)_{b,b'} = \int \varphi_{b'} \varphi_b r dr dz \quad (4.7.26)$$

4.7. Axisymmetric Variational formulation of the 2D Maxwell's equation

Therefore, the discrete equations can be written in the form:

$$M_r \partial_t [E_z] = (D_r^T + M)[B], \quad (4.7.27)$$

$$M_r \partial_t [E_r] = -D_z^T [B], \quad (4.7.28)$$

$$M_r \partial_t [B] = -(D_r + M)[E_z] + D_r [E_r]. \quad (4.7.29)$$

Time scheme

Equations 4.7.27, 4.7.28 and 4.7.29, can be written in the form:

$$M^e \partial_t [E] = K[B], \quad (4.7.30)$$

$$M_r \partial_t [B] = -K^T [E] \quad (4.7.31)$$

where,

$$M^e = \begin{pmatrix} M_r & 0 \\ 0 & M_r \end{pmatrix}, \quad [E] = \begin{pmatrix} [E_z] \\ [E_r] \end{pmatrix}, \quad \text{and} \quad K = \begin{pmatrix} D_r^T + M \\ -D_z^T \end{pmatrix}$$

As in the previous section, we will use the Leap Frog time scheme, which leads to the LF time discretization of order N , given by :

$$\begin{cases} M^e \frac{[E]^{n+1} - [E]^n}{\Delta t} = K_N [B]^{n+\frac{1}{2}} \\ M_r \frac{[B]^{n+\frac{3}{2}} - [B]^{n+\frac{1}{2}}}{\Delta t} = -K_N^T [E]^{n+1} \end{cases}$$

where

$$\begin{cases} K_N = K & , N = 2 \\ K_N = K \left(I - \frac{\Delta t^2}{24} M^{-e} K M_r^{-1} K^T \right) & , N = 4 \end{cases}$$

In figure 4.13, we have plotted the L^2 error norm and computed the convergence order, at $t = 0.01$. We get the correspondent convergence order, *i.e* $p + 1$ for a discrete space based on splines of degree p .

4.7.3 H-rot formulation

In the sequel, we will solve the Maxwell's equation in axisymmetric coordinates, using the **H-rot** formulation, based on the diagram 4.3.9, following the same idea as in the section 4.6. The discretized spaces involved in this case are:

$$V = \mathbf{span}\{N_i^p(x)N_j^p(y), 1 \leq i \leq N_x, 1 \leq j \leq N_y\},$$

$$W_{curl} = \mathbf{span}\left\{ \begin{pmatrix} D_i^p(x)N_j^p(y) \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ N_i^p(x)D_j^p(y) \end{pmatrix}, \quad 1 \leq i \leq N_x, 1 \leq j \leq N_y \right\},$$

$$X = \mathbf{span}\{D_i^p(x)D_j^p(y), 1 \leq i \leq N_x, 1 \leq j \leq N_y\}.$$

As in the case of the **H-div** formulation, we will need to transform carefully the vector functions in order to conserve their properties. In this case,

$$W_{curl} = \mathbf{span}\left\{ \frac{1}{\Delta} \tilde{D}_i^p(\xi) \tilde{N}_j^p(\eta) \begin{pmatrix} \beta_2 \\ -\beta_1 \end{pmatrix}, \frac{1}{\Delta} \tilde{N}_i^p(\xi) \tilde{D}_j^p(\eta) \begin{pmatrix} -\alpha_2 \\ \alpha_1 \end{pmatrix} \right\}. \quad (4.7.32)$$

4.7. Axisymmetric Variational formulation of the 2D Maxwell's equation

After discretization, we get the linear system :

$$\begin{cases} M^W \dot{\mathbf{e}} = K \mathbf{h} \\ M^V \dot{\mathbf{h}} = -K^T \mathbf{e} \end{cases} \quad (4.7.33)$$

where,

$$\begin{aligned} M_{b,b'}^W &= \int_{\Omega} \Psi_b \cdot \Psi_{b'} r dr dz, \\ M_{b,b'}^V &= \int_{\Omega} \varphi_b \varphi_{b'} r dr dz, \\ K_{b,b'} &= \int_{\Omega} \varphi_b \text{rot} \Psi_{b'} r dr dz. \end{aligned}$$

with the usual notation:

$$\begin{aligned} W_{curl} &= \text{span} \{ \Psi_b \}. \\ V &= \text{span} \{ \varphi_b \}. \end{aligned}$$

Numerical results

Let us go back to the test presented in section 4.5.1, with the analytical solutions given by equations 4.5.16, 4.5.17 and 4.5.18. Results are given in figure 4.14. We recover the expected theoretic convergence order.

4.7.4 Remarks

Let us investigate the impact of the variational formulation on the *CFL* number. In tables 4.4 and 4.5, we give the theoretic *CFL* in the following cases:

1. Square domain, in cartesian coordinates, using **H-div** formulation,
2. Square domain, in cartesian coordinates, using **H-rot** formulation,
3. Square domain, in axisymmetric coordinates, using **H-div** formulation,
4. Square domain, in axisymmetric coordinates, using **H-rot** formulation.

We remark that the *CFL* is better when using the **H-rot** formulation.

p = 2	0.8676	1.7335
p = 3	0.5866	1.1644
p = 4	0.4265	0.8516
p = 5	0.3281	0.6588

Table 4.4: Square domain - cartesian coordinates: CFL-LF4 depending on the formulation : (left) using **H-div** formulation, (right) using **H-rot** formulation

4.8. Conclusions and perspectives

p = 2	0.8613	1.2276
p = 3	0.5824	0.8428
p = 4		0.6287
p = 5		0.4941

Table 4.5: Square domain - axisymmetric coordinates: CFL-LF4 depending on the formulation: (left) using **H-div** formulation, (right) using **H-rot** formulation

4.8 Conclusions and perspectives

We presented a new scheme for Maxwell's equation using B-splines functions, which enables as to reduce the number of degrees of freedom thanks to the k-refinement. Our method can be easily used with an integrated CAD system. However, the use of the Isogeometric idea requires to verify the De Rham diagram, so, we can not always use an exact modeling. therefore, we will need to approximate or interpolate domains using either splines or NURBS functions. Another idea, which can be a great help is the use of GB-Splines [75, 76, 22]. GB-Splines are a general basic splines, and verify a closed relation to (4.3.11).

We have constructed new basis functions based on appropriately chosen tensor products of B-spline functions that generate several discrete spaces involved in the numerical solution of Maxwell's equations. These discrete spaces are the same as those introduced by Buffa et al. and proved there to form an exact sequences on which a convergence proof can be based. This new setting provides a simple and efficient way to use B-splines for the solution of Maxwell equations. Due to the discrete exact sequence all the geometric properties satisfied by the Whitney Finite Elements still hold in our context. Moreover we verified that for smooth solutions a lot fewer degrees of freedom are needed for spline Finite Elements than for same order Whitney Finite Elements for the same accuracy. Finally our spline Finite Elements can be naturally constructed on a computational domain defined by NURBS curves or surfaces generated by a CAD system.

In the case of axisymmetric coordinates, we have shown that we can not use the same DeRham sequence as for the cartesian coordinates, in the case of the **H-div** formulation. Finding an appropriate sequence might be very interesting. However, in the case of the **H-rot** formulation, we recover the expected theoretic convergence order.

For some applications, our approach could be extended to an approximation of the electromagnetic fields using GB-Splines. GB-Splines are general basic splines, and verify a closed relation to (4.3.11):

$$G'_{i,k}(t) = \frac{G_{i,k-1}(t)}{c_{i,k-1}} - \frac{G_{i+1,k-1}(t)}{c_{i+1,k-1}}, \quad k \geq 3$$

with

$$G_{i,2}(t) = \begin{cases} \psi_{i,n}^{(n-2)}(t), & t_i^* \leq t \leq t_{i+1}^* \\ \phi_{i+1,n}^{(n-2)}(t), & t_{i+1}^* \leq t \leq t_{i+2}^* \\ 0, & t \notin (t_i^*, t_{i+2}^*) \end{cases}$$

and $c_{i,k-1} = \int_{t_i^*}^{t_{i+k-1}^*} G_{i,k-1}(t) dt$, $\psi_{i,n}^{(n-2)}$, $\phi_{i,n}^{(n-2)}$ are assumed to be strictly monotone on (t_i^*, t_{i+1}^*) and (t_{i+1}^*, t_{i+2}^*) respectively. We can get the classical polynomial spline of order

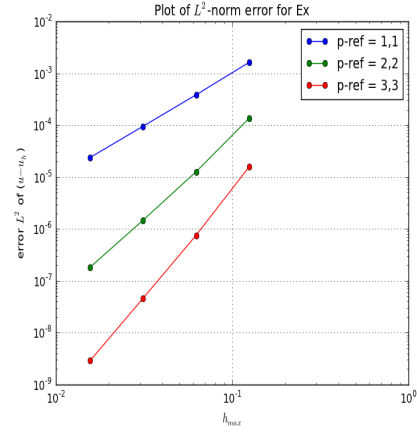
4.8. Conclusions and perspectives

n by taking

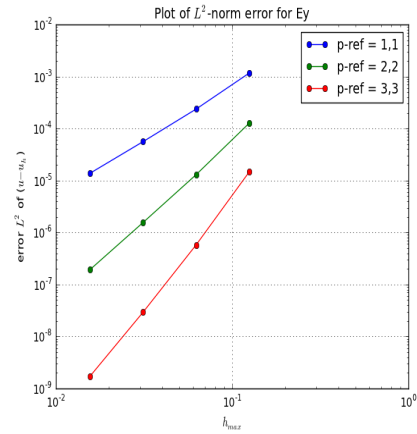
$$\phi_{i,n}(t) = -\frac{(t - t_i^*)^{n-1}}{(n-1)!h_i}, \quad \psi_{i,n}(t) = \frac{(t - t_i^*)^{n-1}}{(n-1)!h_i}, \quad h_i = t_{i+1}^* - t_i^*$$

Thanks to this recurrence formula, which is of the form of 4.3.11, we will be able to do same normalization of the basis which will lead to delete one of the mass matrices. Using such functions, we can perform a *Mass Lumping* technique. In fact, for certain parameters we noticed a diminution of 20% of the non zeros elements of the Mass matrix. This is under development.

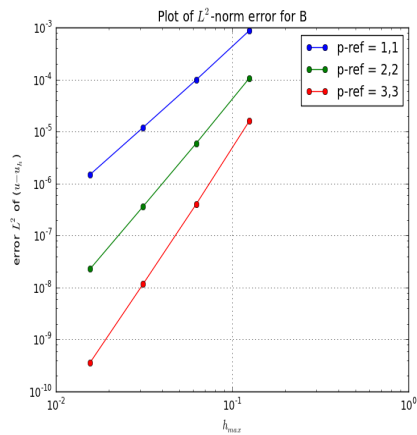
4.8. Conclusions and perspectives



$p = 2, 1$	2.0791	2.0161	2.0038
$p = 3, 2$	3.4231	3.1047	3.0206
$p = 4, 3$	4.3873	4.0394	3.9975



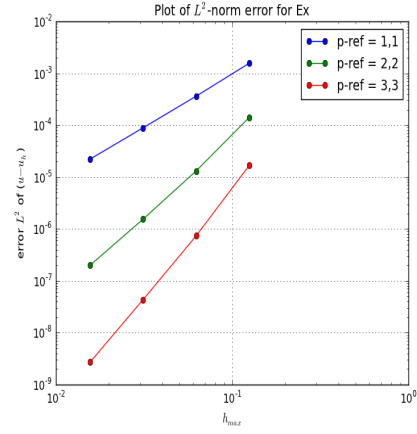
$p = 1, 2$	2.2985	2.0816	2.0209
$p = 2, 3$	3.2882	3.0578	3.0113
$p = 3, 4$	4.6607	4.3011	4.0930



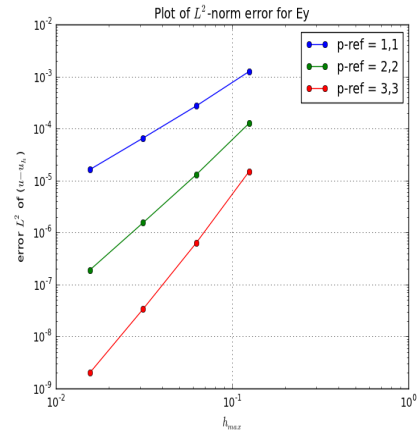
$p = 2, 2$	3.1438	3.0403	3.0104
$p = 3, 3$	4.1581	4.0308	3.9990
$p = 4, 4$	5.3096	5.0972	5.0269

Figure 4.6: Circular wave guide test using the polar mapping and C^0 periodicity-condition : at $t = 0.0$, the L^2 norm error (left) and the convergence order (right) for, 1st line: E^x , 2nd line: E^y , 3rd line: the magnetic field B

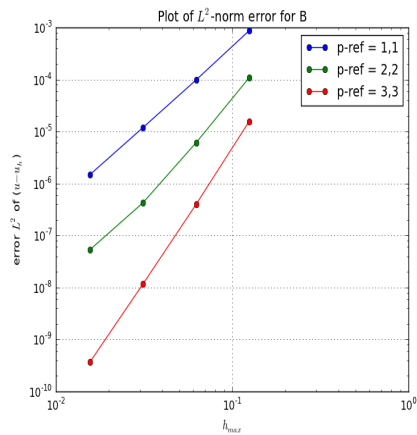
4.8. Conclusions and perspectives



$p = 2, 1$	2.1072	2.0263	2.0077
$p = 3, 2$	3.4210	3.0915	2.9544
$p = 4, 3$	4.5051	4.0865	4.0106



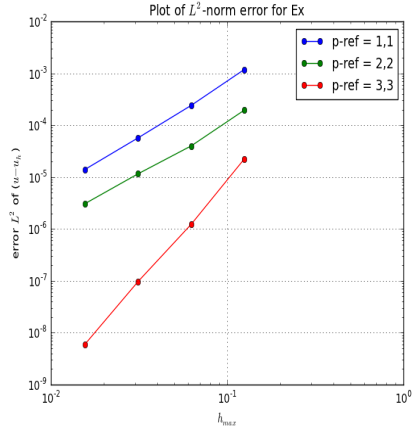
$p = 1, 2$	2.2108	2.0465	2.0089
$p = 2, 3$	3.2951	3.0647	3.0127
$p = 3, 4$	4.5811	4.2198	4.0592



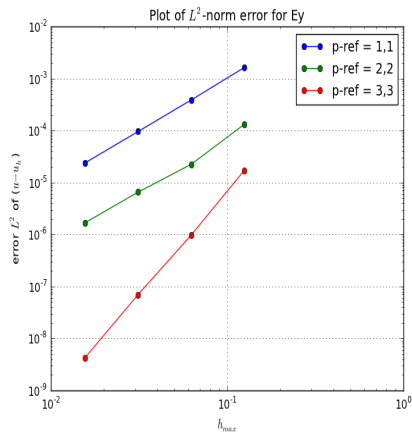
$p = 2, 2$	3.1471	3.0403	3.0097
$p = 3, 3$	4.1526	3.8353	3.0033
$p = 4, 4$	5.3016	5.0838	4.9900

Figure 4.7: Circular wave guide test using the polar mapping and C^0 periodicity-condition : at $t = 0.05$, the L^2 norm error (left) and the convergence order (right) for, 1st line: E^x , 2nd line: E^y , 3rd line: the magnetic field B

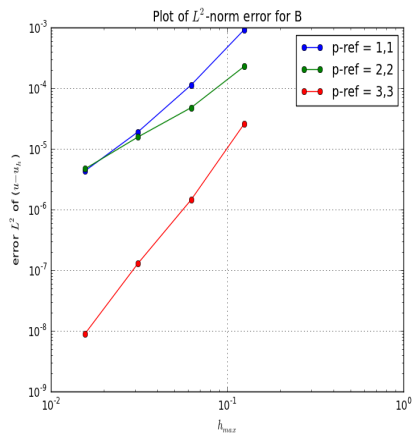
4.8. Conclusions and perspectives



$p = 2, 1$	2.2972	2.0777	2.0196
$p = 3, 2$	2.2890	1.7850	1.9095
$p = 4, 3$	4.1817	3.6617	4.0116



$p = 1, 2$	2.0710	2.0155	2.0034
$p = 2, 3$	2.5530	1.7775	1.9557
$p = 3, 4$	4.1252	3.8068	4.0253



$p = 2, 2$	3.0317	2.5654	2.1401
$p = 3, 3$	2.2594	1.6027	1.7505
$p = 4, 4$	4.1574	3.4791	3.8574

Figure 4.8: Circular wave guide test using the polar mapping and C^0 periodicity-condition : at $t = 0.5$, the L^2 norm error (left) and the convergence order (right) for, 1st line: E^x , 2nd line: E^y , 3rd line: the magnetic field B

4.8. Conclusions and perspectives

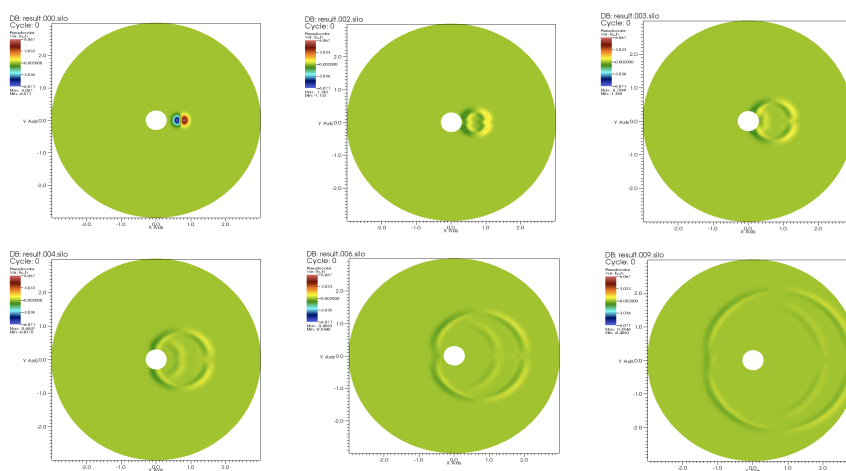
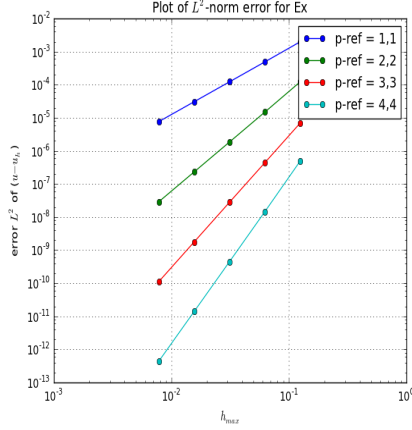
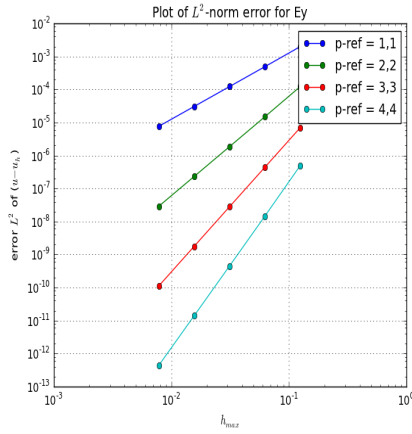


Figure 4.9: Evolution of an electromagnetic wave in circular domain under Silver-Muller boundary condition

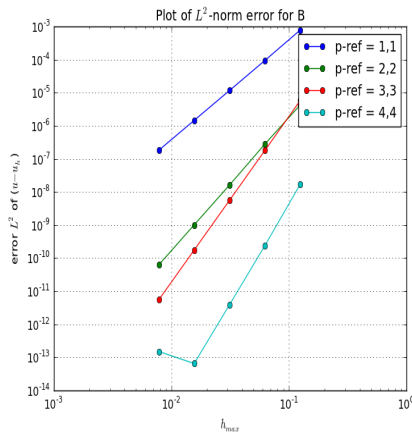
4.8. Conclusions and perspectives



$p = 1, 2$	2.0208	2.0037	2.0003	1.9998
$p = 2, 3$	3.0385	3.0100	3.0022	3.0004
$p = 3, 4$	3.9836	3.9774	3.9848	3.9914
$p = 4, 5$	5.0859	5.0247	5.0063	4.9981



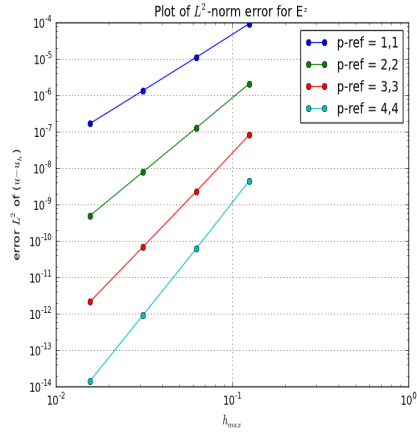
$p = 2, 1$	2.0208	2.0037	2.0003	1.9998
$p = 3, 2$	3.0385	3.0100	3.0022	3.0004
$p = 4, 3$	3.9836	3.9774	3.9848	3.9914
$p = 5, 4$	5.0859	5.0247	5.0063	4.9982



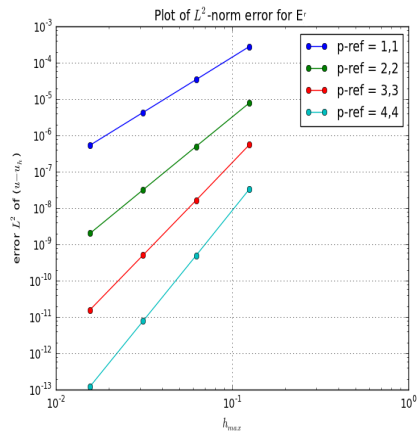
$p = 2, 2$	3.0596	3.0157	3.0032	3.0008
$p = 3, 3$	3.9323	4.1170	4.0002	4.0201
$p = 4, 4$	5.0230	5.0062	5.0016	4.9998
$p = 5, 5$	6.1666	5.9391	5.9089	-1.2096

Figure 4.10: Square wave guide test using H-rot formulation : the L^2 norm error (left) and the convergence order (right) for, 1st line: E^x , 2nd line: E^y , 3rd line: the magnetic field B

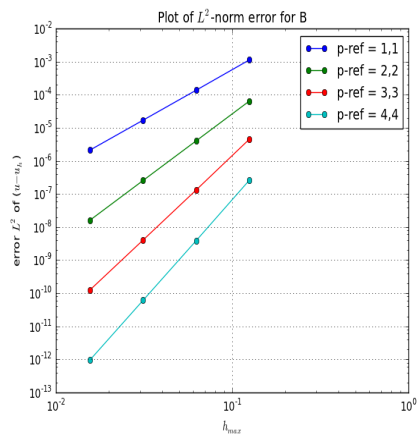
4.8. Conclusions and perspectives



$p = 2, 1$	3.0714	3.0271	3.0114
$p = 3, 2$	4.0372	4.0048	3.9988
$p = 4, 3$	5.1757	5.0420	5.0090
$p = 5, 4$	6.1985	6.0510	6.0117



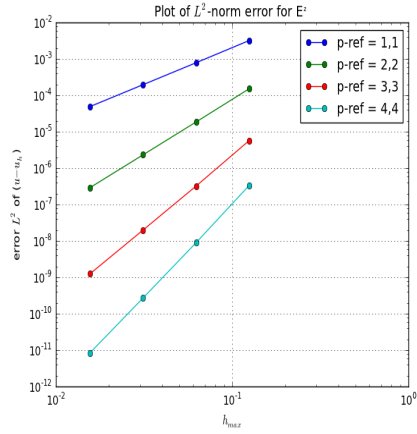
$p = 1, 2$	3.0159	3.0044	3.0011
$p = 2, 3$	3.9630	3.9732	3.9845
$p = 3, 4$	5.0665	5.0179	5.0046
$p = 4, 5$	6.0471	5.9983	5.9925



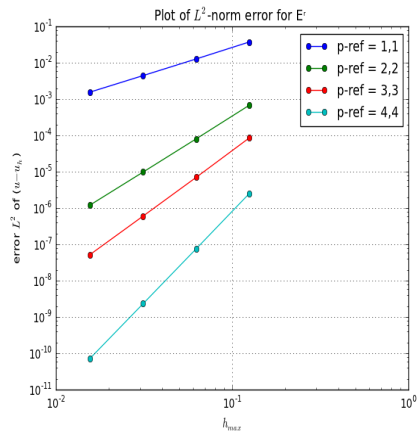
$p = 2, 2$	3.0354	3.0087	3.0018
$p = 3, 3$	3.9845	3.9787	3.9856
$p = 4, 4$	5.0864	5.0257	5.0070
$p = 5, 5$	6.0554	5.9893	5.9843

Figure 4.11: Square wave guide test in axisymmetric coordinates : at $t = 0.0$, the L^2 norm error (left) and the convergence order (right) for, 1st line: E^z , 2nd line: E^r , 3rd line: the magnetic field B

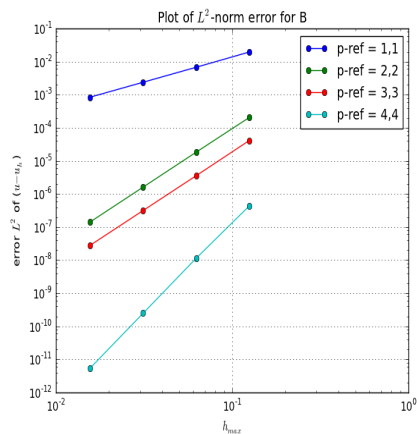
4.8. Conclusions and perspectives



$p = 2, 1$	2.0267	2.0030	1.9992
$p = 3, 2$	3.0286	3.0012	2.9971
$p = 4, 3$	4.1499	4.0170	3.9958
$p = 5, 4$	5.2235	5.0525	5.0107



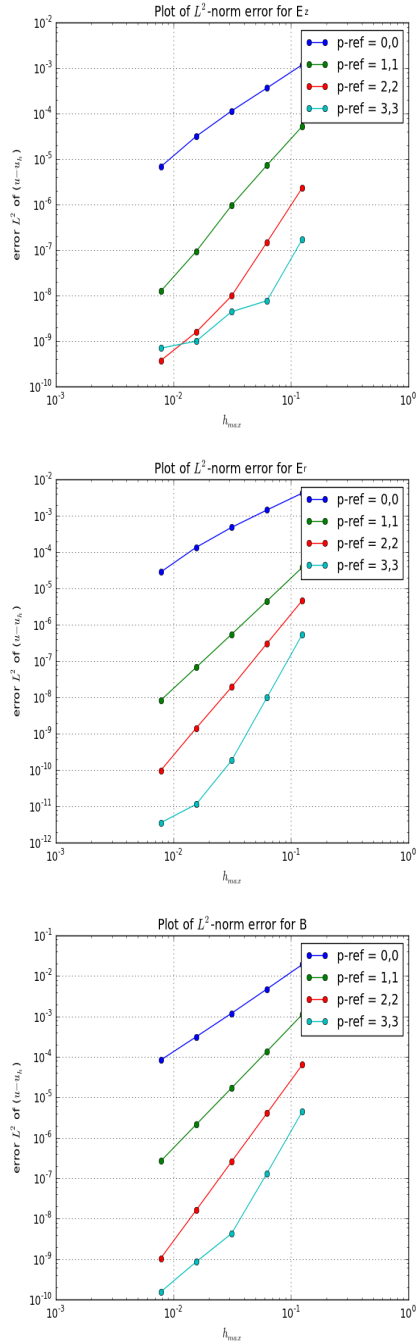
$p = 1, 2$	1.5514	1.5219	1.5100
$p = 2, 3$	3.0716	3.0378	3.0172
$p = 3, 4$	3.6055	3.5411	3.5218
$p = 4, 5$	5.0503	5.0218	5.0049



$p = 2, 2$	1.5278	1.5064	1.5015
$p = 3, 3$	3.4792	3.5101	3.5047
$p = 4, 4$	3.5283	3.4974	3.4992
$p = 5, 5$	5.2508	5.5199	5.5185

Figure 4.12: Square wave guide test in axisymmetric coordinates : the L^2 norm error (left) and the convergence order (right) for, 1st line: E^z , 2nd line: E^r , 3rd line: the magnetic field B

4.8. Conclusions and perspectives



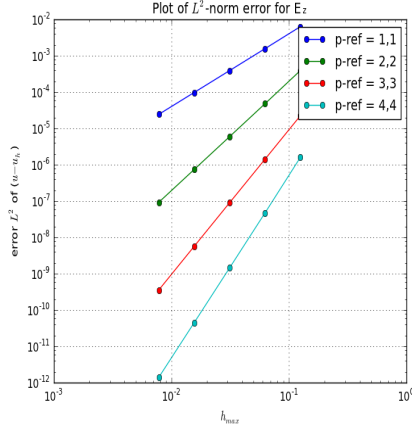
$p = 1, 1$	1.7007	1.6761	1.8550	2.2167
$p = 2, 2$	2.8501	2.9308	3.3615	2.9031
$p = 3, 3$	3.9786	3.9191	2.6482	2.0748
$p = 4, 4$	4.4928	0.8006	2.1500	0.5124

$p = 1, 1$	1.5446	1.5904	1.8248	2.2541
$p = 2, 2$	3.0802	3.0480	3.0227	3.0085
$p = 3, 3$	3.9386	3.9886	3.7554	3.8725
$p = 4, 4$	5.7821	5.7574	4.0066	1.6888

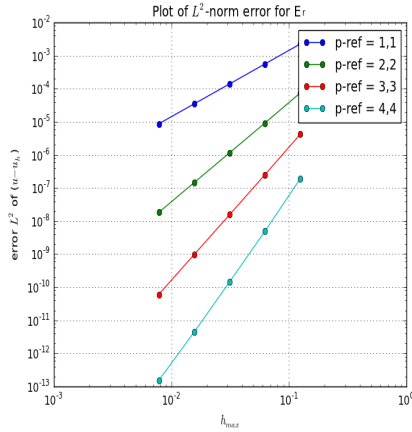
$p = 1, 1$	2.0159	1.9916	1.9322	1.9046
$p = 2, 2$	3.0353	3.0080	3.0001	3.0022
$p = 3, 3$	3.9838	3.9766	3.9861	3.9452
$p = 4, 4$	5.0867	4.9531	2.3205	2.4643

Figure 4.13: Square wave guide test in axisymmetric coordinates (2nd formulation) : the L^2 norm error (left) and the convergence order (right) for, 1st line: E^z , 2nd line: E^r , 3rd line: the magnetic field B

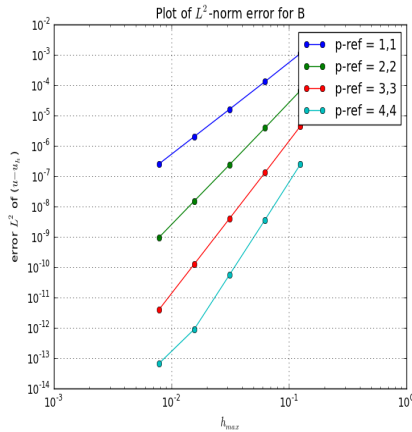
4.8. Conclusions and perspectives



$p = 1, 2$	2.0164	2.0032	2.0003	1.9998
$p = 2, 3$	3.0369	3.0095	3.0022	3.0004
$p = 3, 4$	3.9814	3.9770	3.9847	3.9914
$p = 4, 5$	5.0838	5.0244	5.0063	4.9992



$p = 2, 1$	2.0135	2.0045	2.0005	1.9999
$p = 3, 2$	3.0262	2.9889	2.9895	2.9936
$p = 4, 3$	4.0698	4.0157	4.0018	3.9992
$p = 5, 4$	5.2810	5.0913	5.0336	4.8522



$p = 2, 2$	3.0766	3.0201	3.0031	3.0005
$p = 3, 3$	4.0776	4.0300	3.9901	4.0006
$p = 4, 4$	5.0764	5.0255	5.0065	5.0018
$p = 5, 5$	6.0846	5.9970	5.9844	3.7644

Figure 4.14: Square wave guide test in axisymmetric coordinates, using **H-rot** formulation : the L^2 norm error (left) and the convergence order (right) for, 1st line: E^z , 2nd line: E^r , 3rd line: the magnetic field B

An axisymmetric PIC code based on Isogeometric Analysis

Contents

5.1	Introduction	98
5.1.1	Domain parametrization using Splines/NURBS curves	98
5.2	PIC method for Vlasov equation	98
5.2.1	The PIC Method	99
5.2.2	The equations of motion	100
5.2.3	The Dirac mass with a change of variables	101
5.2.4	Computing J and ρ with a change of variables	102
5.3	Particles emission	102
5.3.1	Short description of a diode	102
5.3.2	Extraction conditions	102
5.4	Numerical results	103
5.4.1	Emission of particles in the diode	103
5.5	Conclusion and perspectives	103

5.1 Introduction

The Vlasov equation describes the evolution of charged particles in an electromagnetic field, which can consist of an applied and a self-consistent field. The latter being computed using Maxwell's equations. Hence, we need to consider the system of the Vlasov-Maxwell equations. Physical problems described by this system are varied and it is necessary to develop methods adapted for each one of them.

We are interested here in the emission of electrons in a diode with hemispherical cathode. The problem is a priori three-dimensional and we can use cylindrical coordinates (z, r, θ) . Moreover this problem is such that the unknowns do not depend on θ so that we can assume 2D axisymmetric geometry.

The IsoPIC code that we describe in this paper is a (z, r) axisymmetric Vlasov-Maxwell solver.

Thanks to symmetry in θ direction, it is enough to give the description of the domain for a section (for example $\theta = 0$). The isogeometric analysis approach will be applied, and we will construct a mapping \mathbf{F} transforming a square into the section.

For the Maxwell equations, we consider in a first stage only the transverse electric mode (denoted by TE). The electric and magnetic field components considered are E_r , E_z and B_θ . To solve them, we use spline finite elements. This has been studied in the chapter 4. The present chapter describes an axisymmetric Vlasov-Maxwell Particle In Cell (PIC) method based on isogeometric analysis.

5.1.1 Domain parametrization using Splines/NURBS curves

The parametrization of the domain by curves defined by *B-splines* and *NURBS* is a recent topic. In fact, it is a very important question to be able to create a surface meshes giving only a description of the boundary in term of curves. An additional properties may be required: in the case of Maxwell's equations, it is well known (see for example [90]) that the mapping must be continuously differentiable, one-to-one and onto map, such that the determinant of the jacobian is of one sign on all the parametric domain (*i.e* patch). For the moment, only few articles discussing the topic provide strategies to build meshes [2, 21, 84, 118] from the description of the boundary, but nothing about additional properties of the corresponding mapping.

The easiest approach, which involves NURBS to describe the quarter of circle of the diode, leads to a mapping which is continuous at the crossing of some nodes and not C^1 , see Fig. 5.1. We thus opted for a version in the spirit of isoparametric analysis using only B-splines. The elements used to describe the boundary improve this description by increasing the degree of splines. A quarter of the circle is constructed by interpolation, by choosing a degree, determined by the user. This can also be done by approximation.

For numerical validation, we will work on a version of the domain, obtained by keeping the control points and weights equal to 1 (in which case the NURBS become B-splines), is presented later. This domain is used for validation, before looking a more advanced version.

5.2 PIC method for Vlasov equation

The main idea of a PIC method is to consider a set of macro-particles, which under electromagnetic fields (Lorentz force) will move (Vlasov equation). Those macro-particles

5.2. PIC method for Vlasov equation

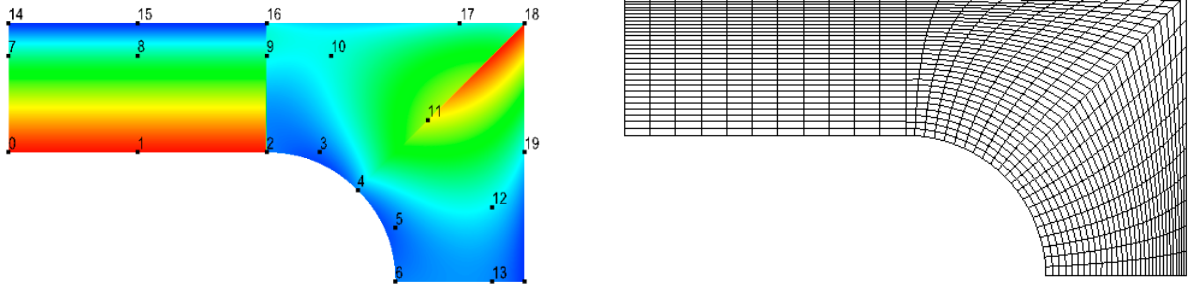


Figure 5.1: Diode: (left) The Jacobian of the mapping and the control points, (right) the mesh.

are in fact a mathematical model of cloud of (micro) particles. Hence, a typical model would be to consider a distribution of those (mini) particles around this macro-particle. As the Maxwell's equations are written in the patch, it would be better to be able to move particles in this parametric domain. The aim of this section, is to write the equations of motion in the patch.

5.2.1 The PIC Method

To solve the Vlasov equation

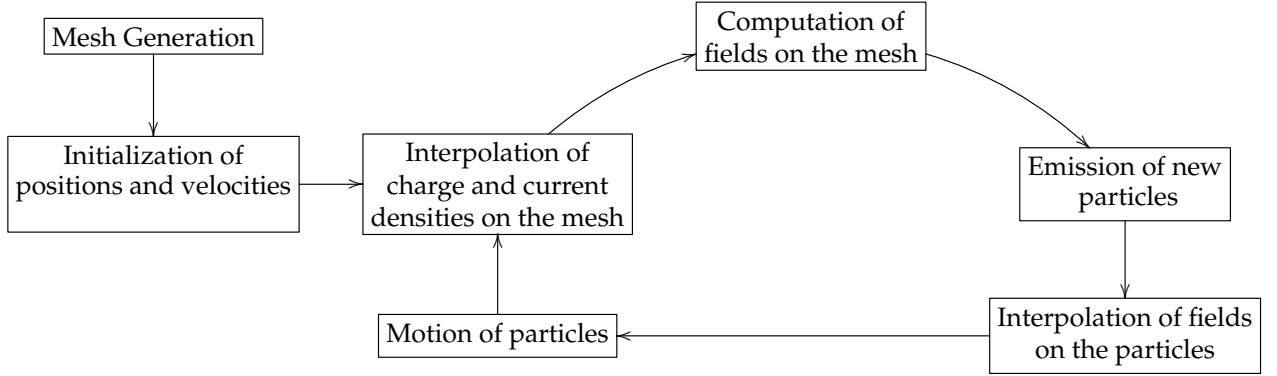
$$\frac{\partial f}{\partial t} + \mathbf{V} \cdot \nabla_X f - (\mathbf{E} + \mathbf{V} \wedge B) \cdot \nabla_V f = 0,$$

we use the PIC method [103], [11]. We consider a set of N macro-particles with \mathbf{X}_k being as the position, \mathbf{V}_k as the velocity, and ω_k as the weight. The particles represent the distribution function f . We approach f by a sum of Dirac functions centered in the positions and in the velocities of particles:

$$f(\mathbf{X}, \mathbf{V}, t) \approx f_N(\mathbf{X}, \mathbf{V}, t) = \sum_{k=1}^N \omega_k \delta(\mathbf{X} - \mathbf{X}_k(t)) \delta(\mathbf{V} - \mathbf{V}_k(t)). \quad (5.2.1)$$

The motion of particles is described by the equations of motion in which the electric and magnetic fields are used. They are obtained by solving the Maxwell equations which involve the charge density ρ and the current density \mathbf{J} . We present the different steps of the PIC method:

5.2. PIC method for Vlasov equation



The most important steps that we develop in the following are:

- the description of equations of motion,
- the computation of charge density and current density,
- the condition for the emission of the particles.

5.2.2 The equations of motion

In our case, we move the particles in the patch, because it is easier to localize them on it. Otherwise, we will need to inverse the mapping for all particles, and at each time step, which is obviously inadmissible. Let us explain how to obtain the equations of motion in generalized coordinates, more details can be found in the appendix C.

The equations of motion in Lagrangian mechanics are the Lagrange equations, also known as the Euler-Lagrange equations. There are available in any coordinate system:

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\mathbf{q}}}(\mathbf{q}, \dot{\mathbf{q}}, t) = \frac{\partial L}{\partial \mathbf{q}}(\mathbf{q}, \dot{\mathbf{q}}, t),$$

where $L(\mathbf{q}, \dot{\mathbf{q}}, t)$ is the Lagrangian.

To write and solve the equations of motion in any coordinate system, we must know the Lagrangian to obtain the Lagrange equations. We start from its expression in cylindrical coordinates, where denoting by \mathbf{A} the vector potential and ϕ the scalar potential, it has the form

$$L(z, r, \theta, \dot{z}, \dot{r}, \dot{\theta}, t) = \frac{1}{2}m(\dot{r}^2 + r^2\dot{\theta}^2 + \dot{z}^2) - e(A_r\dot{r} + A_\theta\dot{\theta} + A_z\dot{z} - \phi(z, r, \theta)).$$

Now, we define the map F which transforms the cylindrical coordinates $F(\xi, \eta, \theta) = (z, r, \theta)$ into a new coordinate system (ξ, η, θ) . It is denoted by $F(\xi, \eta, \theta) = (z, r, \theta)$. But, since we work in 2D axisymmetric geometry, we have $\dot{\theta} = 0$, and so the map is simplified by $F(\xi, \eta) = (z, r)$.

In this coordinate system, the Lagrangian has the form

$$L(\xi, \eta, \theta, \dot{\xi}, \dot{\eta}, \dot{\theta}, t) = \frac{1}{2}m(M_\xi \dot{\xi}^2 + M_\eta \dot{\eta}^2 + 2M_{\xi\eta} \dot{\xi} \dot{\eta}) - e \left(A_\xi \dot{\xi} + A_\eta \dot{\eta} - \phi(\xi, \eta, \theta) \right),$$

with $(A_\xi, A_\eta, A_\theta)$ the components of the vector potential in the coordinate system (ξ, η, θ) and with

$$M_\xi = \left(\frac{\partial r}{\partial \xi} \right)^2 + \left(\frac{\partial z}{\partial \xi} \right)^2, \quad M_\eta = \left(\frac{\partial r}{\partial \eta} \right)^2 + \left(\frac{\partial z}{\partial \eta} \right)^2, \quad M_{\xi\eta} = \frac{\partial r}{\partial \xi} \frac{\partial r}{\partial \eta} + \frac{\partial z}{\partial \xi} \frac{\partial z}{\partial \eta}.$$

5.2. PIC method for Vlasov equation

From this new Lagrangian, we can deduce the equations of motion in the new coordinate system:

$$\begin{aligned} \det(J) \frac{d\dot{\xi}}{dt} + \dot{\xi}^2 K_{\xi,\eta} + \dot{\eta}^2 K_{\eta,\eta} + 2\dot{\eta}\dot{\xi} K_{\eta\xi,\eta} &= -\frac{e}{m \det(J)} (((\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_{\xi}) M_{\eta} - (\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_{\eta}) M_{\xi\eta}) \\ \det(J) \frac{d\dot{\eta}}{dt} - \dot{\xi}^2 K_{\xi,\xi} - \dot{\eta}^2 K_{\eta,\xi} - 2\dot{\eta}\dot{\xi} K_{\xi\eta,\xi} &= -\frac{e}{m \det(J)} (((\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_{\eta}) M_{\xi} - (\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_{\xi}) M_{\xi\eta}) \end{aligned}$$

with

$$K_{\xi,\xi} = H_{\xi} V_{\xi}, \quad K_{\xi,\eta} = H_{\xi} V_{\eta}, \quad K_{\eta,\eta} = H_{\eta} V_{\eta}, \quad K_{\eta,\xi} = H_{\eta} V_{\xi}, \quad K_{\xi\eta,\xi} = H_{\xi\eta} V_{\xi}, \quad K_{\xi\eta,\eta} = H_{\xi\eta} V_{\eta},$$

where

$$H_{\xi} = \begin{pmatrix} \frac{\partial^2 z}{\partial \xi^2} \\ \frac{\partial^2 r}{\partial \xi^2} \end{pmatrix}, \quad H_{\eta} = \begin{pmatrix} \frac{\partial^2 z}{\partial \eta^2} \\ \frac{\partial^2 r}{\partial \eta^2} \end{pmatrix}, \quad H_{\xi\eta} = \begin{pmatrix} \frac{\partial^2 z}{\partial \xi \partial \eta} \\ \frac{\partial^2 r}{\partial \xi \partial \eta} \end{pmatrix}, \quad V_{\xi} = \begin{pmatrix} \frac{\partial r}{\partial \xi} \\ -\frac{\partial z}{\partial \xi} \end{pmatrix}, \quad V_{\eta} = \begin{pmatrix} \frac{\partial r}{\partial \eta} \\ -\frac{\partial z}{\partial \eta} \end{pmatrix}.$$

$\det(J) = \frac{\partial z}{\partial \xi} \frac{\partial r}{\partial \eta} - \frac{\partial z}{\partial \eta} \frac{\partial r}{\partial \xi}$ is the Jacobian of the change of coordinates F , the components of \mathbf{E} , \mathbf{B} are the components of the electric field and the magnetic field in (ξ, η, θ) and $\dot{\mathbf{q}} = \begin{pmatrix} \dot{\xi} \\ \dot{\eta} \end{pmatrix}$.

Numerically, we consider $\dot{\xi}$ and $\dot{\eta}$ as independent variables with $\dot{\xi} = \frac{d\xi}{dt}$ and $\dot{\eta} = \frac{d\eta}{dt}$. We then solve the resulting first order system of ordinary differential equations using the second-order Runge-Kutta method.

5.2.3 The Dirac mass with a change of variables

Let C be a cell of the physical domain, \tilde{C} is such that $\mathbf{F}(\tilde{C}) = C$. The Dirac mass at a point \mathbf{x}_k is denoted by $\delta(\mathbf{x} - \mathbf{x}_k)$. It has the following properties:

$$1 = \int_C \delta(\mathbf{x} - \mathbf{x}_k) d\mathbf{x}, \quad \text{and } g(\mathbf{x}_k) = \int_C \delta(\mathbf{x} - \mathbf{x}_k) g(\mathbf{x}) d\mathbf{x}, \quad \text{for any continuous function } g.$$

Since the integral value does not change when we change the coordinates we can deduce that

$$1 = \int_C \delta(\mathbf{x} - \mathbf{x}_k) d\mathbf{x} = \int_{\tilde{C}} \delta(\mathbf{F}(\xi, \eta) - \mathbf{x}_k) \text{Jac}(\xi, \eta) d\xi d\eta = \int_{\tilde{C}} \frac{\delta(\xi - \xi_k) \delta(\eta - \eta_k)}{\text{Jac}(\xi_k, \eta_k)} \text{Jac}(\xi, \eta) d\xi d\eta,$$

with $\mathbf{F}(\xi_k, \eta_k) = \mathbf{x}_k$ and because by definition $\delta(\mathbf{F}(\xi, \eta) - \mathbf{x}_k) = \frac{\delta(\xi - \xi_k) \delta(\eta - \eta_k)}{\text{Jac}(\xi_k, \eta_k)}$, so

$$g(\mathbf{x}_k) = \int_C \delta(\mathbf{x} - \mathbf{x}_k) g(\mathbf{x}) d\mathbf{x} = \int_{\tilde{C}} g(\mathbf{F}(\xi, \eta)) \delta(\mathbf{F}(\xi, \eta) - \mathbf{x}_k) \text{Jac}(\xi, \eta) d\xi d\eta$$

hence,

$$g(\mathbf{x}_k) = g(\mathbf{F}(\xi_k, \eta_k)). \quad (5.2.2)$$

5.3. Particles emission

5.2.4 Computing J and ρ with a change of variables

In the physical domain, we have $\rho(\mathbf{X}, t) = -\int_{\mathbf{V}} f(\mathbf{X}, \mathbf{V}, t) d\mathbf{V}$, and $\mathbf{J}(\mathbf{X}, t) = -\int_{\mathbf{V}} f(\mathbf{X}, \mathbf{V}, t) \mathbf{V} d\mathbf{V}$. Replacing f by its sum of Dirac function (5.2.1), we have:

$$\rho(\mathbf{X}, t) = -\sum_{k=1}^N \omega_k \delta(\mathbf{X} - \mathbf{X}_k(t)),$$

$$\mathbf{J}(\mathbf{X}, t) = -\sum_{k=1}^N \omega_k \mathbf{V}_k(t) \delta(\mathbf{X} - \mathbf{X}_k(t)).$$

Numerically, we have to compute the integral of these functions in space. It is easier to do it on the patch. With help of previous part and the equality (5.2.2), we deduce, for any $\psi \in (L^2(\Omega))^2$ and $\varphi \in L^2(\Omega)$,

$$\int_C \mathbf{J}(\mathbf{X}, t) \cdot \psi d\mathbf{X} = \int_{\tilde{C}} \sum_k \omega_k \delta(\mathbf{F}(\xi, \eta) - \mathbf{X}_k(t)) \mathbf{V}_k(t) Jac(\xi, \eta) d\xi d\eta = \sum_{k|\mathbf{X}_k(t) \in C} \omega_k \mathbf{V}_k(t) \cdot \psi(\mathbf{X}_k(t)),$$

and

$$\int_C \rho(\mathbf{X}, t) \varphi d\mathbf{X} = \int_{\tilde{C}} \sum_k \omega_k \delta(\mathbf{F}(\xi, \eta) - \mathbf{X}_k(t)) Jac(\xi, \eta) d\xi d\eta = \sum_{k|\mathbf{X}_k(t) \in C} \omega_k \varphi(\mathbf{X}_k(t)),$$

where C is a cell of a physical domain and \tilde{C} a cell in the patch such as $F(\tilde{C}) = C$.

5.3 Particles emission

5.3.1 Short description of a diode

A diode is constituted of two semiconductors: a cathode rich in electrons and an anode which lacks them. If we apply a positive tension at the anode and a negative one at the cathode, such that the created potential drop is greater than a threshold value, electrons are extracted of the cathode and move towards the anode. This phenomena allows an electric current to pass. The movement of electrons near the cathode can create a vicious circle: they increase the potential drop, allowing it to propagate in the diode, and new electrons are extracted and moved.

Numerically, we can impose a negative electric field at the entry of the diode and allow it to propagate. If it is strong enough, it extracts electrons of the cathode, they go towards the anode. Our domain is meshed, so we look at each cell to see if the field satisfies the conditions of the emission of particles.

5.3.2 Extraction conditions

At a given frequency, particles are created in each cell C respecting the following conditions:

- the cell C touches the cathode,
- the normal electric field to the cathode on C is greater than a threshold value.

5.4. Numerical results

Weights of created particles are positive and such that the following relation is respected: $\text{div } \mathbf{E} = \rho$, and in the same time, the normal electric field to the cathode is zero according to the Child-Langmuir law. We integrate this relation on a cell C :

$$\int_C \text{div } \mathbf{E} d\Omega = \int_C \rho d\Omega.$$

The Stokes formula leads to: $\int_C \text{div } \mathbf{E} d\Omega = \int_{\partial C} \mathbf{E} \cdot \mathbf{n} d\gamma$. Besides, the approximation (5.2.1) gives:

$$\int_C \rho d\Omega = \sum_{\text{particle } k \in C} -\omega_k$$

because particles are electrons, their charge is negative but their weight is positive. We obtain the relation

$$\int_{\partial C} \mathbf{E} \cdot \mathbf{n} d\gamma = \int_{\partial C_0} \mathbf{E} \cdot \mathbf{n} d\gamma + \int_{\partial C_1} \mathbf{E} \cdot \mathbf{n} d\gamma = \sum_{\text{particle } k \in C} -\omega_k,$$

where ∂C_0 is the boundary of C touching the cathode and ∂C_1 is the union of other boundaries of C . In order to make $\int_{\partial C_0} \mathbf{E} \cdot \mathbf{n} d\gamma$ vanish, the weight of injected particles is imposed such that we have

$$\int_{\partial C_1} \mathbf{E} \cdot \mathbf{n} d\gamma = \sum_{\text{particle } k \in C} -\omega_k.$$

5.4 Numerical results

5.4.1 Emission of particles in the diode

We studied qualitatively the extraction of particles in the domain representing the diode. We impose at the beginning of the diode a tension that increases linearly until reaching a threshold value, and is then constant. Particles are emitted at the cathode under the conditions of extraction described previously. We take the time step $dt = 1.0 \times 10^{-2}$ and perform 25 000 iterations. We represent the electric field E_r and the particles at the time $T = 25, 50, 75, 100, 125, 150, 175$ and 200 in the figure 5.2.

5.5 Conclusion and perspectives

We have developed the IsoPIC code which solves the Vlasov-Maxwell equations in 2D axisymmetric geometry. It is based on isogeometric analysis. It is applied to emit electrons in a diode with a hemispherical cathode, which we approximated using splines. The solution of the Vlasov equation is performed with a PIC method by moving the particles emitted at the cathode.

The Maxwell solver in axisymmetric geometry gives good results. The theoretical orders of convergence are checked on a square and the fields evolve correctly in the diode. Qualitatively, we also have good results for the particles. The extraction seems to be good

5.5. Conclusion and perspectives

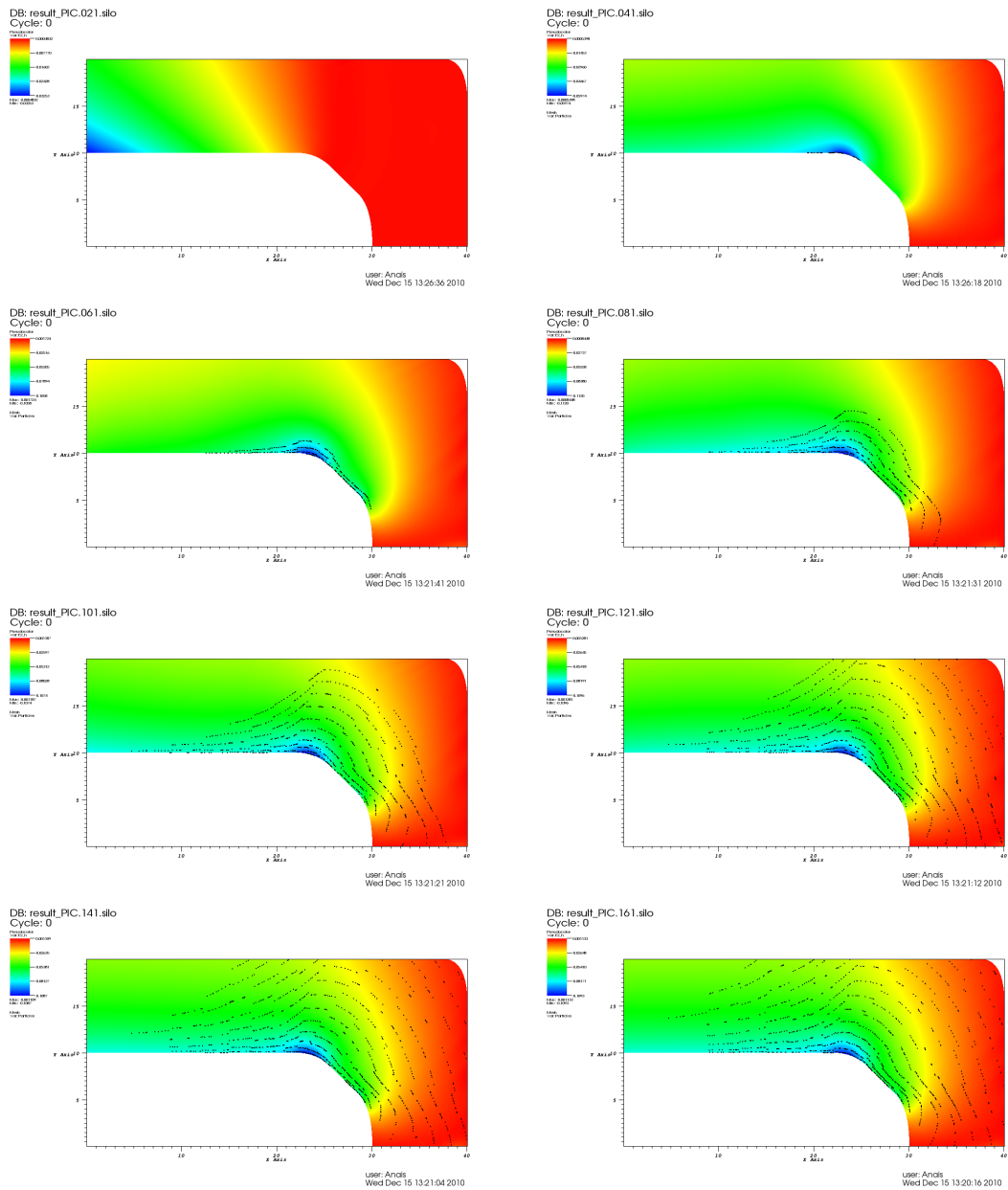


Figure 5.2: Electric field and electrons at time $T=25$, $T=50$ (first line), $T=75$, $T=100$ (second line), $T=125$, $T=150$ (third line), $T=175$ and $T=200$ (last line).

but we have to add a confinement for which the TM mode, we have not implemented up to now, is needed.

We noticed that the conjugate gradient method is not good for our problem because the mass matrix of the electric field in axisymmetric geometry is not well conditioned so this algorithm does not converge fast enough. We did not notice this problem in cartesian coordinates where the matrices are better conditioned. So we used the direct solver Pastix, which has the advantage to be faster than iterative solvers for this kind of problems in 2D.

The equations of motion are written and implemented in a general coordinate system.

5.5. Conclusion and perspectives

We have tested them in particular cases and we have compared the results with polar coordinates. The use of generalized coordinates is a choice for this study, despite the bigger calculation time. We would like to test later other possibilities, like cartesian coordinates with a mapping/inverse mapping to handle the mesh.

This work needs to be continued addressing the following topics. On the one hand, we can rewrite the Maxwell equations in order to not have to invert the mass matrix of the electric field. On the other hand, we have to compute a Poisson solver, in order to consider more general cases. In fact, if there are particles at initial time (which is not the case in this study), we have to solve the Poisson equation initially. Then, the conservation of charge is obtained thanks to the exact De Rham sequence. Finally, we can parallelize the Maxwell solver and the motion of particles using GPU (Graphics Processing Unit), to reduce computation time. This kind of parallelization is indeed very efficient in our costly computations, which do not require a lot of memory.

Application to Semi-Lagrangian schemes

Contents

6.1	2D Vlasov	109
6.1.1	Physical model: the paraxial beam	109
6.1.2	The ISOLOSS code	110
6.2	Complex geometry using parametric surfaces	113
6.2.1	General framework	113
6.2.2	Analytic mapping	115
6.2.3	Bézier patches	115
6.3	Algorithms	116
6.3.1	Inverse mapping for Bézier patches	117
6.3.2	Reducing delays for patch finding	119
6.3.3	Velocity integrals	120
6.4	Results	120
6.4.1	Geometry settings and experimental results	120
6.4.2	Performance issues	122

Introduction

A tokamak is a toroidal device for plasma confinement in which a strong toroidal magnetic field is imposed by a system of external coils, while a poloidal magnetic field is generated by a strong toroidal current flowing through the plasma. The sum of these toroidal and poloidal fields results in a helical geometry of the magnetic field lines. In the region we are interested in, which is the confined plasma in the core of the tokamak, the magnetic field lines are closed and are wrapped around closed surfaces. These so-called magnetic surfaces are nested around the plasma center and can usually be considered axisymmetric and described by a constant section around the torus. The shape of the magnetic surfaces has a strong impact on the physics involved in the confinement of the plasma [4]. Therefore, in order to accurately describe the transport processes in a tokamak plasma, a fine description of the geometry of magnetic surfaces is mandatory.

The strong magnetic field in a tokamak means that the motion of particles will be restricted in the direction perpendicular to the magnetic field lines, but free along this (so-called parallel) direction. Moreover, the large perturbations along the magnetic field lines play an important role in transport processes and must be included in the model. Thus, it appears that even small discretization errors can corrupt numerical results, for instance by causing a parallel heat flux to leak into the transverse direction [47]. These issues are particularly critical when modeling nonlinear phenomena such as turbulence, which is usually studied in the fusion community through the development of gyrokinetic codes [45]. These first-principle codes solve a coupled Vlasov-Poisson system in 5 dimensions (3 dimensions in real space and 2 in velocity space). As a first approach, most of these codes adopted a simplified description of the geometry of magnetic surfaces, for instance using basic polar coordinates, which implies that realistic tokamak geometries were not taken into account. More recently, an ongoing effort has been initiated to include more realistic geometries. As a long-term objective, we expect to design a new Vlasov solver for the gyrokinetic code GYSELA [53] which would extend the code's capability to perform simulations in complex tokamak geometry while providing a fine description of this geometry.

Classical FEM method with straight lines describing the edges are not adapted to the complex geometry of magnetic surfaces in a tokamak, where curved elements appear necessary for an accurate description of these surfaces. In this context, the isoparametric and isogeometric frameworks provide a method to produce curved elements adapted to any given geometry, and to create a PDE solver using these elements. Compared to a more standard description by metric tensors, the isoparametric/isogeometric frameworks: 1) give more flexibility to generate and refine the computational mesh, 2) introduce a rich set of computationally cheap parametric functions to build the mesh and to map quantities between configuration space and parameter space, 3) provide the so called *k*-refinement, a strategy that can increase the regularity of functions through the mesh's interfaces, reducing the number of degrees of freedom, 4) allow a simple modification of the boundary by repositioning the control points.

In order to advance towards the long-term objective of designing a gyrokinetic Vlasov solver in complex geometry, we describe in this paper a feasibility study of isoparametric analysis in a simplified setting. The Vlasov solver is implemented on a reduced phase space (1D in space and 1D in velocity), instead of the 5D phase space of gyrokinetic codes. This reduced model is directly relevant for the application expected in GYSELA, as isoparametric meshes would be used – at least as a first step – to generate the 2D meshes

describing magnetic surfaces in the poloidal plane. We consider a Vlasov-Poisson problem modeling the focusing of a heavy ion beam in an axisymmetric geometry around its optical axis. The advantage of this reduced model is that numerical experiments can be performed in a small-sized phase space domain using simple Dirichlet boundary conditions. Since the solution remains very localized, one can easily investigate different mesh geometries without damaging numerical results. Moreover, as is the case in gyrokinetic simulations, small-scale filamentation develops in phase-space, which must be correctly captured by the solver independently of the chosen mesh. The aim of this paper is to validate numerically the Vlasov solver for different underlying meshes, as well as to measure the impact of the chosen mesh on computational costs.

In section 6.1, we describe the two-dimensional Vlasov-Poisson test case considered, as well as the numerical methods used for a standard cartesian mesh. The framework for isogeometric analysis and the chosen computational grids are introduced in section 6.2. The specific algorithms needed to design a Vlasov solver on an isoparametric surface are presented in section 6.3. The results in terms of accuracy and computational cost for the different geometries considered are detailed in section 6.4. A conclusion follows.

6.1 2D Vlasov

6.1.1 Physical model: the paraxial beam

Our reference test case is the study of beam focusing using the paraxial Vlasov-Poisson model[42]. This model is common in accelerator physics and describes the propagation of a particle beam along a linear optical axis. While the beam is considered in steady-state, the propagation velocity in the direction z of the optical axis is assumed constant (thus z is formally replaced by time in the equations). We also assume that the beam is symmetric around its optical axis. Therefore, we solve the Vlasov-Poisson system in cylindrical geometry with the radius r as the only space coordinate and v_r the velocity in the radial direction. In order to avoid issues with boundary conditions around $r = 0$, we consider a symmetric domain in r with the condition $f(-r, v_r) = f(r, -v_r)$ where f is the particle distribution function in phase-space.

Following the normalization in [105], we solve the Vlasov equation coupled with a Poisson equation

$$\partial_t f + v_r \partial_r f + F(t, r) \partial_v f(t, r, v_r) = 0 \quad (6.1.1)$$

$$\frac{1}{r} \partial_r (r E_{self}(t, r)) = \rho(t, r) = \int_{-\infty}^{\infty} f(t, r, v_r) dv_r \quad (6.1.2)$$

where $F(t, r)$ is the applied force. It contains contributions from both the externally applied (E_{app}) and the self-applied (E_{self}) fields:

$$F(t, r) = K_{self} E_{self}(t, r) + K_{app} E_{app}(t, r) .$$

where K_{app} and K_{self} are constants. We consider the case where a proton beam is focused by applying a periodic external electric field with the following normalized expression

$$E_{app}(t, r) = -\frac{1}{2} \{1 + \cos(2\pi t)\}^2 r \quad (6.1.3)$$

6.1. 2D Vlasov

As an initial distribution function, we will use the semi-gaussian (gaussian in velocity, localized in space) formulation

$$f_0(r, v_r) = \frac{N_0}{\sqrt{2\pi}a^2v_{th}} e^{-\frac{v_r^2}{2v_{th}^2}} \text{ for } |r| < a \quad (6.1.4)$$

$$= 0 \quad \text{elsewhere} \quad (6.1.5)$$

where N_0 is the total number of particles, a is the initial beam radius and v_{th} is the thermal velocity.

6.1.2 The ISOLOSS code

In order to solve the 2D Vlasov-Poisson system in complex geometry, we have used the LOSS code [25], extending it for the paraxial beam test case and for various geometries to a new version named ISOLOSS. The ISOLOSS code uses a semi-Lagrangian numerical scheme[106], which will be presented in section 6.1.2. The time integration is performed using a predictor-corrector scheme, described in section 6.1.2.

Semi-Lagrangian method

Two types of methods are most commonly used to solve the Vlasov-Poisson system. On the one hand, Lagrangian (or Particle-In-Cell) methods[12] discretize the distribution function into a finite number of macro-particles. The evolution of these macro-particles then follows the characteristics of the Vlasov equation, while a grid is necessary only in real space in order to solve the Poisson equation. Lagrangian methods allow for efficient computations with a numerical cost that can be tuned depending on the expected accuracy. The main drawback of such methods is the numerical noise due to the sampling of the distribution function, which requires complex noise reduction methods (see for instance [73] for state-of-the-art noise reduction techniques in gyrokinetic simulations). On the other hand, Eulerian methods solve the Vlasov equation on a fixed grid in phase-space using finite differences, finite volumes or spectral methods to discretize the operators in the Vlasov equation. The key issue for Eulerian methods is that the discretization of the operators leads to numerical dissipation.

The semi-Lagrangian method is a mix between the Lagrangian and Eulerian methods, which tries to eliminate the main drawbacks of each method. This method was first developed for meteorological studies (see [107] for a review) and was more recently adapted to plasma simulations [106]. In order to avoid the statistical noise observed in Lagrangian codes, a fixed (Eulerian) grid in phase-space is used. On the other hand, to avoid numerical dissipation and take advantage of the conservation of the distribution function along trajectories by the Vlasov equation, the characteristics of the equation are used to compute the time evolution of the distribution function. The basic algorithm for the backward semi-Lagrangian method [106] used in LOSS is described in figure 6.1. For every grid point at a given time step, the characteristic curves are integrated backward to find the value of the distribution function at the foot of the characteristic. As this point is hardly ever on the grid, an interpolation must be performed to compute the value of the distribution function. It has been shown [43, 10] that cubic spline interpolation provides a good compromise between accuracy (low diffusivity) and numerical cost. Details of this method will be presented in section 6.1.2.

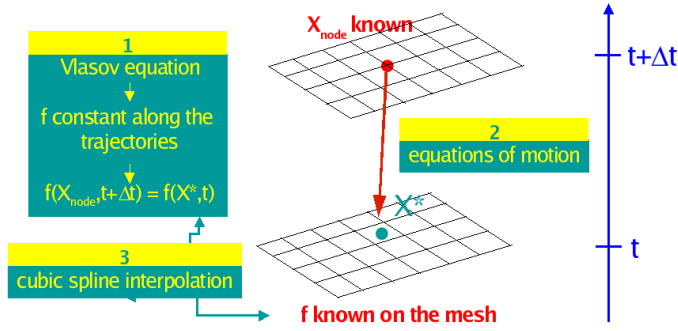


Figure 6.1: Basic algorithm for the Semi-Lagrangian method (figure from [52])

Predictor-corrector scheme

The time integration of the Vlasov-Poisson system is performed using a second-order in time predictor-corrector numerical scheme. Given the distribution function $f(t^n, r, v_r)$ at time t^n we perform the following sequence to compute f at $t^{n+1} = t^n + \Delta t$:

1. Computation at time t^n of the charge density ρ^n (different methods are considered for this computation, depending on the choice of geometry, see 6.3.3 for details)
2. Computation of the self-consistent electric field E^n by solving the Poisson equation (6.1.2) with $\rho = \rho^n$
3. For each grid point (r, v_r) : backward advection of $\Delta t/2$ to compute the foot of the characteristic (r^*, v_r^*)
4. Interpolation on the 2D grid using local cubic splines to compute $f(t^n, r^*, v_r^*)$, which yields the value for $\tilde{f}(t^{n+1/2}, r, v_r)$ due to the conservation of the distribution function along the characteristics
5. Computation of the density $\tilde{\rho}^{n+1/2}$ and electric field $\tilde{E}^{n+1/2}$
6. For each grid point (r, v_r) : backward advection (and interpolation) of Δt using the fields at $t^{n+1/2}$ to compute $f(t^{n+1}, r, v_r)$ from the values $f(t^n)$

This numerical scheme is similar to a second-order Runge-Kutta method.

Following the characteristics

In the semi-Lagrangian method, one has to compute characteristic curves between two consecutive time steps. The foot of one characteristic, denoted previously (r^*, v_r^*) , is found by approximating an advection term using the velocity and the force fields. Several procedures can be used in order to find this foot, such as Taylor expansion [52] or fixed-point iterations. We choose a predictor-corrector scheme that gives an advection term approximated with an error of second order in space (see [24] for detailed description).

2D cubic spline interpolation

The interpolation of the distribution function on the grid in phase-space is one of the main challenges of the semi-Lagrangian method. The method used for this interpolation

6.1. 2D Vlasov

should limit dissipation while being numerically efficient, as it will be performed at each time step and for every grid point. In the LOSS and ISOLOSS codes, this interpolation is performed using two-dimensional cubic splines [31, 106]. In this section, we will first present the method for interpolation with cubic splines in one dimension, then extend this method to two dimensions using a tensor product of cubic spline basis.

In one dimension, consider a function $g(x)$ defined for $x \in [x_0, x_N]$. This function g is projected on a basis of cubic splines:

$$g(x) \simeq cs(x) = \sum_{\nu=-1}^{N+1} \eta_{\nu} B_{\nu}(x) \quad (6.1.6)$$

where B_{ν} are the cubic B-splines and the coefficients η_{ν} are the unknown spline coefficients. The interpolation of the function g by cs on the $N + 1$ points of the domain leads to $N + 1$ equations ($cs(x_i) = g(x_i)$ for $i = 0, \dots, N$) for the $N + 3$ spline coefficients. As the linear system is undetermined, we add two constraints on the derivatives at the boundaries

$$g'(x_i) \simeq cs'(x_i) = \frac{1}{2h} (\eta_{i+1} - \eta_{i-1}) \quad \text{for } i = 0 \text{ and } i = N \quad (6.1.7)$$

where h is the uniform cell size defined by $h = x_{i+1} - x_i$. Thus the spline cs interpolating g at the grid points and verifying the boundary conditions is uniquely defined. The vector of spline coefficients $\eta = [\eta_{-1}, \dots, \eta_{N+1}]^T$ is obtained by solving the linear system $A\eta = G$ where

$$G = [g'(x_0), g(x_0), \dots, g(x_N), g'(x_N)]^T \quad (6.1.8)$$

and A is the following $(N + 3) \times (N + 3)$ matrix:

$$A = \frac{1}{6} \begin{pmatrix} -3/h & 0 & 3/h & 0 & \dots & 0 \\ 1 & 4 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 \\ 0 & \dots & 0 & -3/h & 0 & 3/h \end{pmatrix} \quad (6.1.9)$$

Note that the Hermite boundary conditions can, depending on the simulation domain, be replaced by periodic boundary conditions. It modifies the linear system that needs to be solved but not the general method. The LU decomposition of matrix A is easily computed using Gauss elimination. For a given vector G , the spline coefficients are then computed in two steps by solving successively the lower triangular matrix system $L\mu = G$ and the upper triangular matrix system $U\eta = \mu$.

Extending this method from one dimension to two dimensions is achieved by considering the tensor product of two cubic spline bases. Consider a function $g(x, y)$ defined for $(x, y) \in [x_0, x_{N_x}] \times [y_0, y_{N_y}]$. The interpolating spline becomes:

$$g(x, y) \simeq cs(x, y) = \sum_{\nu=-1}^{N_x+1} \sum_{\beta=-1}^{N_y+1} \eta_{\nu, \beta} B_{\nu}(x) B_{\beta}(y) \quad (6.1.10)$$

where the unknowns are the $(N_x + 3) \times (N_y + 3)$ coefficients $\eta_{\nu, \beta}$. We first solve the following system for each value of $j = 0, \dots, N_y$

$$cs(x, y_j) = \sum_{\nu=-1}^{N_x+1} \gamma_{\nu}(y_j) B_{\nu}(x) \quad \text{where} \quad \gamma_{\nu}(y_j) \equiv \sum_{\beta=-1}^{N_y+1} \eta_{\nu, \beta} B_{\beta}(y) \quad (6.1.11)$$

6.2. Complex geometry using parametric surfaces

For each value of j , we obtain $N_x + 1$ interpolation conditions. Imposing values of the derivatives at the boundaries in x leads to $N_y + 1$ linear systems of the type $A\gamma_\nu(y_j) = G$ where A is the matrix (6.1.9) and the vector G is similar to expression (6.1.8). These $N_y + 1$ systems of size $(N_x + 3) \times (N_x + 3)$ are solved using LU decomposition as described for one dimensional interpolation, yielding the solutions $\gamma_\nu(y_j)$ for $\nu = -1, \dots, N_x + 1$ and $j = 0, \dots, N_y$.

To compute the coefficients $\eta_{\nu,\beta}$ in equation (6.1.10), we still need to solve equation (6.1.11) for $\nu = -1, \dots, N_x + 1$. For a given value of ν , the systems previously solved give us $N_y + 1$ interpolation conditions, while we need to solve for $N_y + 3$ unknowns. We force the derivatives at the boundaries in y to complete the system, which implies that we need to compute $\gamma'_\nu(y_0)$ and $\gamma'_\nu(y_{N_y})$. This means solving two additional linear systems of the type $A\gamma'_\nu(y_j) = \partial_y G$ with $j = 0$ and $j = N_y$, where the vector G is defined by derivatives of $g(x, y)$ and A is the matrix in (6.1.9). Finally, for each value of ν , we solve the system $A\eta_{\nu,\beta} = \Gamma_{\nu,\beta}$ with $\Gamma_{\nu,\beta} = [\gamma'_\nu(y_0), \gamma_\nu(y_0), \dots, \gamma_\nu(y_{N_y}), \gamma'_\nu(y_{N_y})]^T$, once again using LU decomposition to obtain the spline coefficients $\eta_{\nu,\beta}$.

To estimate the computational cost of two-dimensional cubic spline interpolations, we need to count the number and size of the linear systems solved by one LU decomposition. To compute all the coefficients, we need to solve:

- $N_y + 1$ systems of size $(N_x + 3) \times (N_x + 3)$
- 2 systems of size $(N_x + 3) \times (N_x + 3)$
- $N_x + 3$ systems of size $(N_y + 3) \times (N_y + 3)$

Considering that the resolution through LU decomposition of a linear system of size N requires $\mathcal{O}(N)$ operations, the global cost of two-dimensional cubic spline interpolation is of the order $\mathcal{O}(N_x N_y)$ operations. The cubic B-splines have compact support basis, the 1D interpolation at a given position requires only the combination of 4 coefficients with 4 basis functions. Let us assume a uniform grid spacing Δx from x_{min} to x_{max} , one has

$$cs(x) = \sum_{\nu=\lfloor \frac{x-x_{min}}{\Delta x} \rfloor - 1}^{\lfloor \frac{x-x_{min}}{\Delta x} \rfloor + 2} \eta_\nu B_\nu(x) . \quad (6.1.12)$$

In a two dimensional setting, the interpolation uses 16 spline coefficients combined with 8 basis functions, leading to a reasonable cost of several tens of floating point operations. In the sequel, $\text{Interp}(g, x, y)$ denotes the 2D interpolation of function g taken at position (x, y) .

6.2 Complex geometry using parametric surfaces

In the following, the symbol t will no longer be used as the time variable but as a parameter in the spatial parametric equations.

6.2.1 General framework

The need of an accurate representation of the geometry is not an exclusive matter of a specific application domain but is quite common in scientific computing. *Diffeomorphisms*

6.2. Complex geometry using parametric surfaces

associated to finite elements have proved to be very useful to deal with complex geometries. The idea is to perform a shape transformation in order to map a computational domain, for example a rectangular grid, to a potentially complex geometrical domain. At a given mesh resolution, a well chosen diffeomorphism adapted to the geometry can reduce numerical errors.

Diffeomorphisms and mapping techniques provide methods to go from a reference coordinate system (e.g. $(s, t) \in [0, 1]^2$) to a physical coordinate system (e.g. $(x, y) \in \Omega$). To do so, one has to choose a *shape function* or a *mapping* to go from the reference system to the physical system. A common example is the use of a polar mapping to map a rectangular grid onto a circular domain. In our context, we choose to work on a uniform grid in the reference coordinate system. Such techniques have also been used in the context of moving grids for the semi-Lagrangian method [104].

In this context, the use of B-splines and NURBS may provide a simple and powerful tool. Since their introduction in the 1960s, B-splines (then NURBS) were almost exclusively used in the CAD community. Recently, Hughes [67] has made the link between the CAD community and the simulation one. The isogeometric analysis gives the user the ability to approach numerical solutions of partial differential equations in the same space used to describe the domain. Isoparametric analysis is generally attributed to Taig [109] and Irons [70]. The idea is to approximate the domain using shape functions. Those functions are then used to approximate the solution of partial differential equations. In fact, when one uses high-order elements for solving a partial differential equation in a domain with curved boundaries, it is essential to include some way of approximating the boundary conditions accurately. The use of isoparametric elements is an efficient way which involves a general piecewise-polynomial change of variables \mathbf{F} in the definition of the discrete spaces. Recalling the classical formulation of the FEM, we use an affine transformation to map a given element K_e to the reference element K . The element basis functions φ_e in K_e are then related to the reference basis functions φ by the relation $\varphi_e(\mathbf{x}) = \varphi(\mathbf{F}^{-1}(\mathbf{x}))$. Elements where the mapping \mathbf{F} comes from the same finite element space are called *isoparametric*. One cost to pay is that this chain rule will be needed for each derivative evaluation. Notice that we must be careful to obtain mappings that have regular jacobians [20, 81]. When the shape functions are B-splines or NURBS, and when the domain is exactly modeled using those shape functions, Isoparametric analysis becomes Isogeometric analysis. Recalling that we can model exactly all conics using NURBS.

One of the most interesting aspects of B-splines is that once the domain is described by B-splines, several strategies exist to refine the mesh:

- by inserting new knots. This is the h-refinement, it is the equivalent of mesh refinement of the classical finite element method.
- by elevating the B-spline degree. This is the p-refinement, it is the equivalent of using higher finite element order in the classical FEM.
- by increasing / decreasing the multiplicity of inserted knots. This is the k -refinement. This new strategy does not have any equivalent in the classical FEM.

One of the most interesting points of such mappings is that the evaluation for a particular point depends only on a finite (and quite small, depending on the spline degree) number of data, which makes the algorithms strongly local. In figure 6.2, one can see that a curve lying between two (interpolating) controls points is uniquely determined by

6.2. Complex geometry using parametric surfaces

the control points defining the corresponding control polygon. This local dependence is very useful as one can change the shape of a curve by only changing the position of some control points (see figure 6.2).

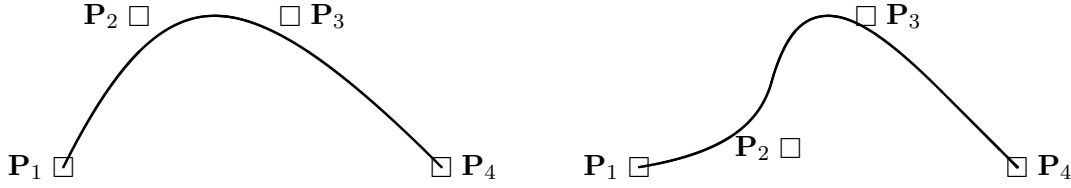


Figure 6.2: A Bézier curve after repositioning the control point P_2

The inconvenient of B-spline mappings is the inverse problem: given a point in the physical domain, what is the corresponding parametric point? Iterative algorithms to perform this inverse mapping will be described in section 6.3.1. Another idea is to use an analytic inverse, whenever it is possible, for example in the case of a polar grid (see section 6.2.2).

A main idea included in this tool set, is to be able to deal with some class of spatial domains, described by B-splines. The aforementioned description can be coupled with a Semi-Lagrangian method to solve the Vlasov equation. The resulting code, that we have named ISOLOSS, will be described in the sequel of this paper. The present work focuses on two isoparametric meshes: an oval mesh with an analytic inverse mapping (section 6.2.2) and a mesh defined by Bézier elements (section 6.2.3), which are a specific type of NURBS or B-splines.

6.2.2 Analytic mapping

As a first step, we consider the case where the mapping is performed by a fully analytic diffeomorphism. This diffeomorphism maps a rectangle $(s, t) \in [0, 1] \times [0, 2\pi[$ (in the reference space) to an ellipse in the physical space. The following parametric equations are taken as a mapping function:

$$x(s, t) = r_{max} \times s \times \cos(t) \quad y(s, t) = v_{max} \times s \times \sin(t)$$

The inverse mapping is also given by analytical expressions:

$$s(x, y) = \sqrt{\frac{y^2}{v_{max}^2} + \frac{x^2}{r_{max}^2}} \quad t(x, y) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0 \text{ and } y \geq 0 \\ \arctan\left(\frac{y}{x}\right) + 2\pi & \text{if } x > 0 \text{ and } y < 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ \frac{3\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \end{cases}$$

The computational cost of moving forward and backward from the reference space to the physical space is quite low. A direct or inverse mapping represents a fraction of the cost induced by one 2D interpolation. Therefore, as we shall see in the numerical experiments, the overhead in the ISOLOSS code is small compared to the original solution that does not require a diffeomorphism.

6.2.3 Bézier patches

A Bézier surface (also known as Bézier patch) of order (n, m) is defined by several control points $(\mathbf{k}_{i,j})_{i \in [0,n], j \in [0,m]}$. It maps the unit square $[0, 1]^2$ into a surface embedded within

6.3. Algorithms

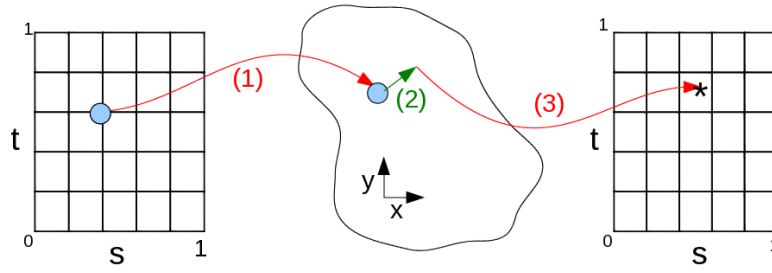


Figure 6.3: The Semi-Lagrangian method in complex geometry: (1) map the position in the reference space into physical space, (2) follow the characteristic backwards in physical space, (3) map the obtained position back in the reference space to perform the interpolation

a space of the same dimensionality as the control points. In our application, we take the control points $\mathbf{k}_{i,j}$ in \mathbb{R}^2 in order to represent a 2D geometry. A Bézier surface is parametric and the equations describing it depend on parameters that are not explicitly part of the geometry. Hence, a point P of coordinates (s, t) on the patch is localized at the following position in the physical space:

$$\mathbf{P}(s, t) = \sum_{i=0}^n \sum_{j=0}^m \mathcal{B}_i^n(s) \mathcal{B}_j^m(t) \mathbf{k}_{i,j} \quad \text{with} \quad \mathcal{B}_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}$$

$\mathcal{B}_i^n(u)$ are known as the Bernstein basis of polynomials of degree n . As a first step, we decide to use biquadratic Bézier patches.

6.3 Algorithms

The Semi-Lagrangian method, as described in section 6.1.2, must be adapted to the formalism of parametric surfaces introduced in section 6.2. In the approach proposed in this paper, the choice has been made to keep the expressions of the advection equations in the physical space, rather than rewriting these equations in the reference space [8]. Therefore, it is necessary to move backward and forward between the reference space and the physical space. Figure 6.3 outlines the adapted Semi-Lagrangian method when using parametric surfaces. From a given position (s, t) in the reference space, we map the corresponding position in physical space. The characteristics are then followed backwards in physical space, and the foot of the characteristics must be mapped back into the reference space before performing the cubic spline interpolation. The main issue in this algorithm is the inverse mapping of positions from physical space to reference space. This transformation has to be performed for each grid point during the advection step. There is no generic analytical solution for curved elements and this operation can be numerically costly. Several solutions have been proposed in the computer graphics community to design fast inverse mappings [46, 89]. Algorithms to perform the inverse mapping from physical space to reference space are presented in section 6.3.1.

Another issue when solving the Vlasov-Poisson system on an isoparametric mesh is the computation of velocity integrals to obtain the density, which is needed to solve the Poisson equation. Whereas such computation is trivial for a cartesian grid in position and velocity, it requires a special treatment for curved elements, which will be presented in section 6.3.3.

6.3.1 Inverse mapping for Bézier patches

Existing solutions

Several algorithms have been proposed to calculate the inverse mapping for Bézier patches [46]. A common approach is to use a multivariate Newton iteration scheme (also called Newton-Raphson scheme). This method uses an iterative process to approach one root of a set of equations. For an adequately suited set of equations, this procedure starts with the position in physical space and gives the position in the reference space within a few iterations. The convergence is quadratic and the number of significant digits double after each iteration. The Newton scheme has an intrinsic problem: it may fail to converge. As the convergence depends on the gradient, we can expect Newton algorithms to fail for domains with singularities.

A method called Bézier clipping [89] has been introduced to guarantee systematic convergence. Iteratively, the algorithm builds smaller nested Bézier subpatches. These Bézier subpatches keep the target point inside them. The parameter domain that surrounds the target point is thus iteratively reduced until it becomes sufficiently small that one can approximate easily the inverse mapping in averaging the control points of the subpatch. This strategy involves much more computations than the Newton-Raphson method.

The inverse mapping algorithm we need is based on the same techniques used for computing the points at which a ray intersects a rational Bézier patch in the computer graphics community. In this work we only investigate Newton and clipping schemes. Some alternative methods (such as interval Newton iteration) have been described in [111].

The inverse mapping function used in the sequel is denoted $\text{Imap}(r, v_r)$. If the input point (r, v_r) is in the computational domain then one has $\text{Imap}(r, v_r) = (s, t) \in [0, 1]^2$.

Description of the Newton algorithm

The inverse mapping problem can be written as:

$$\mathbf{P}(s^*, t^*) = \sum_{i=0}^n \sum_{j=0}^m \mathcal{B}_i^n(s^*) \mathcal{B}_j^m(t^*) \mathbf{k}_{i,j} = (x^*, y^*)$$

where s_* and t_* are the unknown parameters in reference space. We apply Newton's method to solve this nonlinear system. The equation can be rewritten as

$$f(s^*, t^*) = 0 \quad \text{with} \quad f(s, t) = P(s, t) - \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}.$$

The Taylor series of $f(s, t)$ around the point $f(s^*, t^*)$ where f is equal to zero is given by

$$f(s^*, t^*) = f(s, t) + J(s, t) \begin{pmatrix} s^* - s \\ t^* - t \end{pmatrix} + \dots, \quad \text{with} \quad J = \begin{pmatrix} \frac{\partial x^*}{\partial s} & \frac{\partial x^*}{\partial t} \\ \frac{\partial y^*}{\partial s} & \frac{\partial y^*}{\partial t} \end{pmatrix}.$$

At first order we get $\begin{pmatrix} s^* \\ t^* \end{pmatrix} = \begin{pmatrix} s \\ t \end{pmatrix} - J^{-1}(s, t)[f(s^*, t^*) - f(s, t)]$.

The Newton iteration is then $\begin{pmatrix} s^{k+1} \\ t^{k+1} \end{pmatrix} = \begin{pmatrix} s^k \\ t^k \end{pmatrix} - J^{-1}(s^k, t^k)[f(s^{k+1}, t^{k+1}) - f(s^k, t^k)]$.

The inverse of the Jacobian matrix J can be accurately computed in the Bézier formulation, using derivatives of the Bernstein polynomials. The iterative algorithm continues until one of the following three criteria is met:

6.3. Algorithms

1. the L^2 norm of $\left\| \begin{matrix} s^{k+1} - s^k \\ t^{k+1} - t^k \end{matrix} \right\|$ is lower than a threshold,
2. the number of iteration is larger than a predefined maximum number,
3. the position (s, t) is no longer in the Bézier patch.

The Newton algorithm needs to evaluate surface points as well as partial derivatives for given parameter values (s, t) . In order to reduce the number of computations in the Newton kernel, factorizations are performed between the computations of $f(s^k, t^k)$, $\frac{\partial P}{\partial s}(s^k, t^k)$ and $\frac{\partial P}{\partial t}(s^k, t^k)$.

Description of the clipping algorithm

The clipping algorithm uses properties associated to Bézier patches:

- *Convex hull property*: a Bézier surface lies completely within the convex hull of its control points.
- *Subdivision Algorithm for Bézier Curves*: using the DeCasteljau algorithm, one can write a quick algorithm to subdivide a Bézier patch into two Bézier subpatches (see for instance [41]).

Let us describe briefly the different steps of the Bézier clipping (for more details, see [89, 115, 82, 102]). As an input, we have a Bézier patch with its control points $C_{i,j}$ (position in parameter space $(s_{i,j}, t_{i,j})$, position in space $(x_{i,j}, y_{i,j})$). The idea is to reduce the interval of possible values of $s \in [0, 1]$ to a smaller interval that contains the target point P . As a first step, one tries to approximate the intersection of the Bézier patch with a line \mathbb{L} in the direction of the t -axis that goes through P . Suppose that we find a way to know that the intersection of this line \mathbb{L} with the Bézier patch contains only values of s in the interval $[s_{min}, s_{max}]$, one can then deduce immediately that the s coordinates of P is also in the same interval.

To find such an interval, the convex hull property can be used as follows: the intersection of the Bézier patch with \mathbb{L} has to be included in the intersection of the convex hull (defined by the control points $C_{i,j}$) with \mathbb{L} . Computing the intersection of the convex hull with \mathbb{L} is a non trivial procedure, which we present here. First, one defines a line \mathbb{L} almost perpendicular to the s -axis that traverses the target point P . Then, the $d_{i,j}$ distance of each control point $C_{i,j}$ to this line is computed. Because of the perpendicular property of line \mathbb{L} , one can expect that control points at $s = 0$ will be often negative and these at $s = 1$ will be often positive. A new Bézier patch is build with a set built with the following control points $(s_{i,j}, d_{i,j})$. This patch represents a projection that simplifies the intersection task. Suppose the convex hull of these new control points intersects the s -axis at two points s_{min} and s_{max} , such that $0 \leq s_{min} \leq s_{max} \leq 1$. Then we know the intersection points of the Bézier curve and s -axis will be inside $[s_{min}, s_{max}]$. We subdivide the Bézier patch into three patches, with s ranging in $[0, s_{min}]$, $[s_{min}, s_{max}]$, and $[s_{max}, 1]$ respectively. We know that the intervals $[0, s_{min}]$ and $[s_{max}, 1]$ can be safely discarded. Next, we can iterate the whole procedure in the other direction (t) on the new Bézier patch restricted to $s \in [s_{min}, s_{max}]$.

Finally, the algorithm loops onto the described process until the intervals for each parameter are small enough. The proof of convergence of the algorithm is based on the

6.3. Algorithms

following fact: at each iteration one subtracts parameter ranges which are guaranteed not to include the target point.

A sketch of the nested Bézier patches generated during the clipping are depicted in figure 6.4 for an input point located at $(2, 3)$.

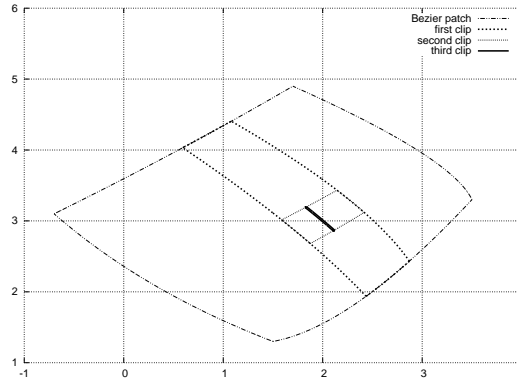


Figure 6.4: Nested Bézier patches generated during clipping algorithm

6.3.2 Reducing delays for patch finding

In order to achieve good performance, it is crucial to reduce the total number of inverse mapping calculations to the minimum. In theory, when applying the Bézier clipping algorithm for inverse mapping, we must iterate over all the Bézier patches until finding the one which actually contains the point we are trying to map. A useful strategy (common in the computer graphics community) consists in subdividing space into more manageable and smaller chunks. We choose to build a collection of axis-aligned bounding boxes around each Bézier patch. Due to the convex hull property of Bézier surfaces, the control points allow us to determine an axis-aligned bounding box in physical space for each patch. For a given point in physical space, if it does not belong to a given bounding box, then it is certain that the point is not inside the associated Bézier patch. One can take advantage of this property to restrict the search of the inverse mapping procedure to a small set of Bézier patches, typically between one and four potential patches. This strategy shortens the time needed to find the patch in which the target point lies.

In order to further improve this strategy, we perform some work in a preprocessing stage to remove costs during inverse mapping computations [82]. Initially, the entire physical domain is bounded by a global bounding box and this box is cut into voxels of equal size. The size of the voxels is chosen such that the volume of a voxel is significantly smaller than the volume of the average bounding box of a Bézier surface. Inside each voxel, a list of the Bézier patches is stored. Given an input point lying in one voxel, this list gives all Bézier patches which could encompass the input point. The list is built based on the axis-aligned bounding boxes previously computed.

Finally, the procedure adopted to obtain the inverse mapping of an input point using Bézier clipping is the following: 1) read the voxel where the list of potential Bézier patches is stored, 2) try applying the clipping algorithm inside each patch of the list and stop as soon as the input point belongs to one of the patches 3) return the localization (s^*, t^*) given by the clipping algorithm.

6.4. Results

In all the test cases considered in this work, where the meshes do not present any singularity, the Newton-Raphson algorithm always converges. Both methods for inverse mapping have been implemented, but the computational cost of the Bézier clipping algorithm is greater by at least one order of magnitude. Therefore, we have used the Newton-Raphson algorithm for the results presented in section 6.4. More generally, a good compromise can be found by using the faster Newton-Raphson algorithm, with the “safer” Bézier clipping algorithm as a backup solution when the Newton scheme does not converge, for instance when the grid presents singularities.

6.3.3 Velocity integrals

The density $\rho(r)$ must be obtained in order to solve the Poisson equation. This density is computed as an integral over the velocity direction, which is not trivial when using a curved grid in phase-space. Indeed, we need a numerical quadrature for approximating the integral over a curved domain represented by isoparametric elements. The inputs available for this computation are the values of the distribution function at the control points positions, and also the spline coefficients used for the interpolation. We choose to approximate the integrals using the trapezoidal rule coupled with a spline interpolation. Let us define the following series $r_q = r_{min} + q\Delta r$ for $q = 0, N_r - 1$ and $v_k = v_{min} + k\Delta v$ for $k = 0, N_v - 1$. The sampled ρ integrals and associated computations are

$$\begin{aligned} \rho(r_q) &= \Delta v \sum_{k=0}^{N_v-1} f(r_q, v_k), \quad (s_{q,k}^*, t_{q,k}^*) = \text{Imap}(r_q, v_k), \\ f(r_q, v_k) &= \text{Interp}(f, s_{q,k}^*, t_{q,k}^*) = \sum_{\nu=\lfloor \frac{s_{q,k}^*}{\Delta s} \rfloor - 1}^{\lfloor \frac{s_{q,k}^*}{\Delta s} \rfloor + 2} \sum_{\beta=\lfloor \frac{t_{q,k}^*}{\Delta t} \rfloor - 1}^{\lfloor \frac{t_{q,k}^*}{\Delta t} \rfloor + 2} \eta_{\nu,\beta} B_\nu(s_{q,k}^*) B_\beta(t_{q,k}^*), \\ \rho(r_q) &= \Delta v \sum_{k=0}^{N_v-1} \sum_{\nu=\lfloor \frac{s_{q,k}^*}{\Delta s} \rfloor - 1}^{\lfloor \frac{s_{q,k}^*}{\Delta s} \rfloor + 2} \sum_{\beta=\lfloor \frac{t_{q,k}^*}{\Delta t} \rfloor - 1}^{\lfloor \frac{t_{q,k}^*}{\Delta t} \rfloor + 2} \eta_{\nu,\beta} B_\nu(s_{q,k}^*) B_\beta(t_{q,k}^*). \end{aligned}$$

where Δt and Δs are the grid sizes in the reference space. Thus, the ρ vector can be expressed as the result of a matrix-vector multiplication $\rho = M\eta$ where η represents the values of the distribution function. Furthermore, the matrix M depends only on the $B_\nu(s_{q,k}^*) B_\beta(t_{q,k}^*)$ products that are available at the beginning of the simulation and remain fixed over time. So, we precompute the matrix M initially and perform sparse matrix-vector multiplication each time we need the density ρ . Let us remark that the size of M barely depends on the number N_v of samples along the v direction. To improve the accuracy of the integrals, one simply needs to perform a bigger precomputation to get the matrix M .

6.4 Results

6.4.1 Geometry settings and experimental results

We consider the evolution of a non stationary beam with the following dimensionless parameters:

6.4. Results

$$N_0 = 2\pi, a = 1.83271471003, v_{th} = 0.0727518214392, K_{self} = 1., K_{app} = 0.5 .$$

In our simulations, the beam is transported over 163 lattice periods, with 100 time steps per period. The underlying uniform 2D grid used for the simulation consists in 2^{18} points covering entirely or partially the spatial domain $(r, v_r) \in [-6, 6] \times [-2.5, 2.5]$. We intend to compare the speed and accuracy on three geometries:

1. *Original LOSS*: no diffeomorphism is employed and this case corresponds to the initial simulator configuration. The computational grid is rectangular and its size is 512×512 .
2. *Analytical mapping*: an analytic shaping function (see section 6.2.2) maps an oval in phase space to a reference domain $[0, 1] \times [0, 1]$. The reference domain has a uniform grid mesh of size 1024×256 with a larger number of points in the angle direction. Especially in this configuration, the periodicity along the angle direction has to be taken into account.
3. *Bezier mapping*: a set of Bézier patches (see section 6.2.3) are built; the shape of the computational domain is in-between the shapes of the previous oval and rectangle configurations. The reference domain has a 512×512 size.

The different computational grids are sketched in figure 6.5 where an undersampling has been applied in order to improve readability.

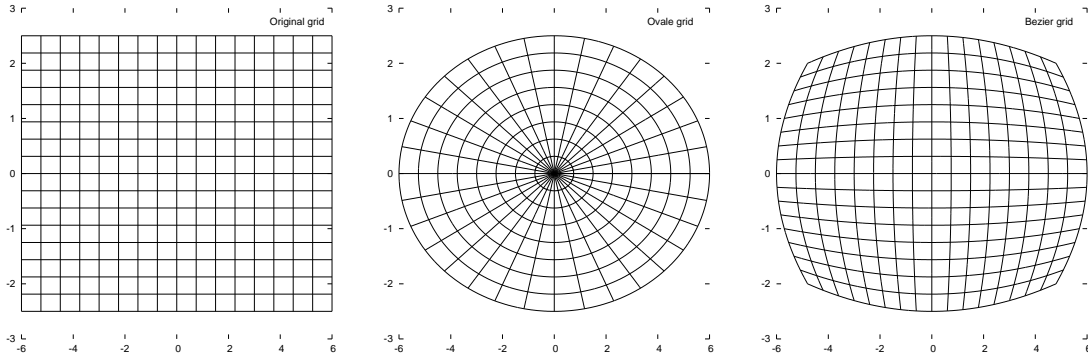


Figure 6.5: Three computational grids are tested for the paraxial beam problem: (a) regular cartesian grid, (b) oval grid associated to an analytical inverse mapping, (c) grid defined by Bézier elements

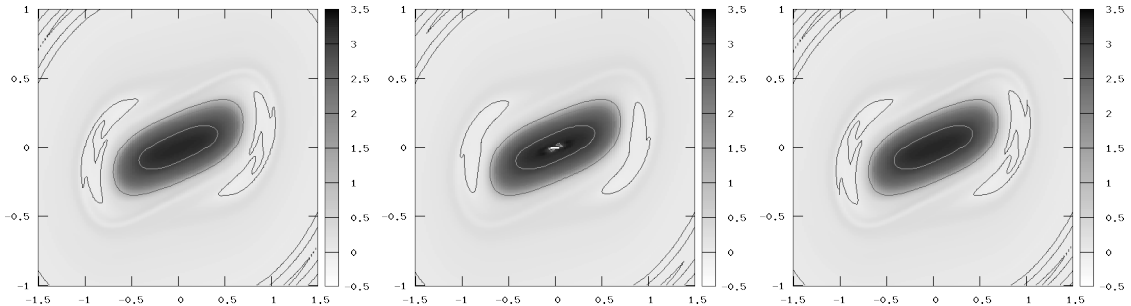


Figure 6.6: Final state of the distribution function in phase space in the three configurations

The final state of the distribution function is shown in figure 6.6 for the different computational grids. For each configuration, the simulator manages to follow the evolution

6.4. Results

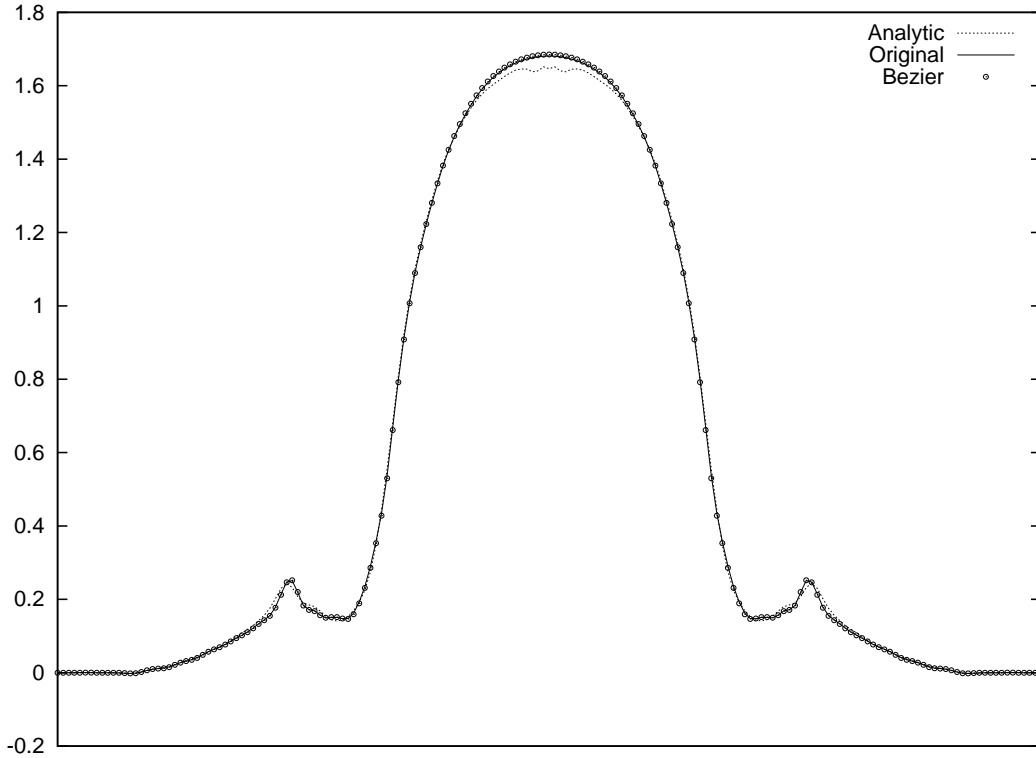


Figure 6.7: Final state of the density function $\rho(r)$ in the three configurations

of filamentation. The distribution functions are quite similar, but the analytic mapping exhibits some clear differences with the two other settings. The strong anisotropy of the oval mapping in the center of the domain clearly triggers numerical problems.

These problems are also illustrated in figure 6.7 where the final state of density $\rho(r)$ is presented in the three geometries. The Bézier and the original curves almost overlay, but the analytic curve has a different behavior at the peak of the density and on the lateral small shoulders. The R_{rms} quantity is displayed in figure 6.8, it measures the effective beam focalisation. The three configurations give nearly identical R_{rms} curves over time. Finally, we conclude that diffeomorphisms do not decrease accuracy directly, but care must be taken in order to avoid unjustified anisotropy.

6.4.2 Performance issues

Performance of the different geometry settings has been investigated on a desktop computer. A 3.0 GHz Intel Core 2 Duo E8400 processor was used to run numerical experiments on only one core. The settings of the runs were exactly the same as in the previous subsection. In Table 6.1, the timings of each part of the code for one simulation of 16384 time steps are gathered. The `Field solve` column sums time used to compute the density ρ density and the self-consistent field. The `Spline coeff.` column corresponds to calls to the LU solver described in section 6.1.2. The `Advection` column comprises the trajectories computation to find the origins of the characteristics and the interpolation costs. The `Total` column sums the first columns, ignoring the preprocessing phase of the simulation and the diagnostics. The field solver and ρ calculation take much more time when a mapping is employed. This is due to the fact that without mapping the

6.4. Results

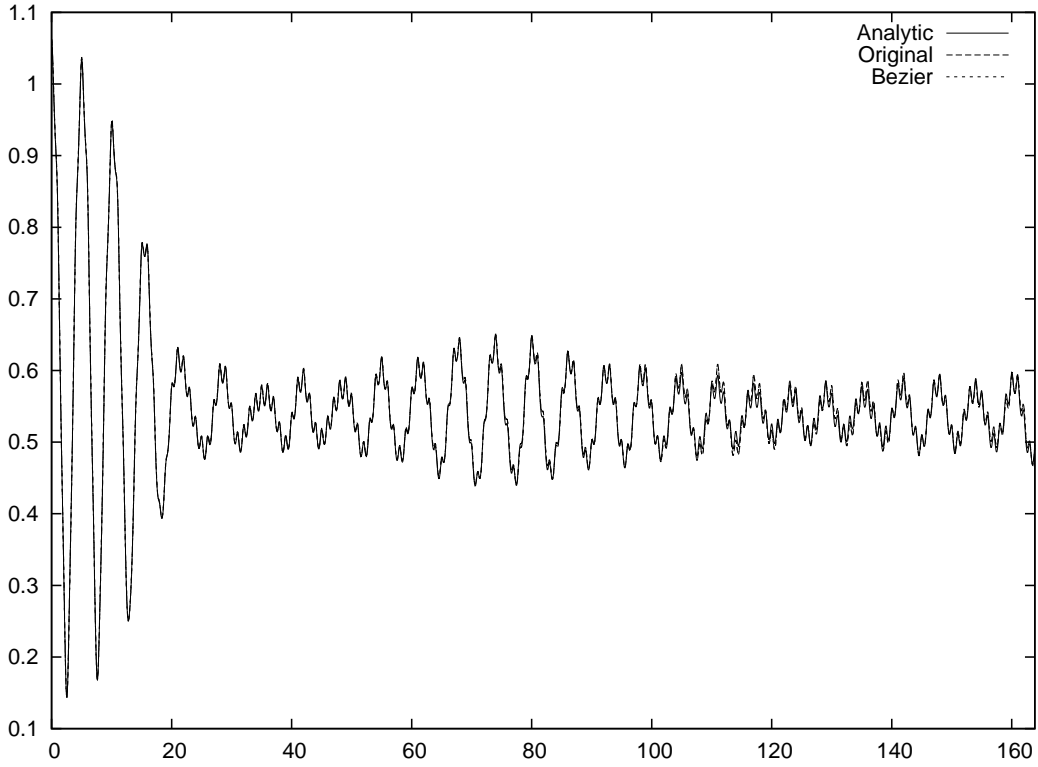


Figure 6.8: Evolution in time of the R_{rms} quantity in the three geometry configurations

	Field solve	Spline coeff.	Advection	Total
Original LOSS	2.3	16.4	78	97
Analytical mapping	10.1	18.4	115	143
Bezier mapping	9.2	17.3	275	301

Table 6.1: Timing results (seconds) for three geometry settings and simulation over 1024 time steps

computation $\rho(r)$ is very light, as it is only a sum of $f(r, v)$ along dimension v ; whereas with mapping the strategy explained in subsection 6.3.3 is used. The computation of spline coefficients takes nearly the same amount of time in each case. The computation costs are almost identical and cache effects are the critical factor that modulates slightly the costs of computing spline coefficients. Concerning the advection step, the cost of the inverse mapping explains the observed differences in computation time. The analytic inverse mapping involves a 50% time increase compared to the original LOSS, and Bézier inverse mapping leads a 250% time increase.

If we look forward to the application of this technique in the GYSELA code, this overhead must be rescaled. The 2D interpolations represents approximately 5% of the execution time in a GYSELA run. A rough estimate of the overhead induced by implementing the Bézier setting in the GYSELA Vlasov solver is thus 13% (with the simple calculation $[275/78 - 1] \times 5\%$). Other overheads should also be considered for the field solver using the upgraded geometry setting, but no simple estimate of this cost can be provided for the moment.

Conclusion

The Semi-Lagrangian scheme combined with isoparametric analysis successfully solves the Vlasov equation on a reduced beam test case. Small-scale filamentations in phase-space are well captured by the new solver. Quantitative analysis shows that only minor issues in accuracy are caused by the mesh geometry. For a parametric Bézier patch, the overhead in terms of computational cost is about 200% on the whole simulation run when compared to the original version. Although this figure may appear quite large, it could in fact be reasonable if the advantages brought by a more accurate description of the domain geometry outweigh its numerical cost. This should be the case for applications with strong geometrical constraints on the computational mesh, such as the gyrokinetic code GYSELA.

Simulation of 2D reduced MHD

Contents

7.1	Introduction	126
7.2	Anisotropic Diffusion	126
7.2.1	Introduction	126
7.2.2	The choice of the grid	126
7.2.3	Evolution of a Gaussian pulse	127
7.2.4	Conclusions	128
7.3	MHD equilibrium	130
7.3.1	Equilibrium in the absence of toroidal flux	130
7.3.2	Equilibrium with toroidal flux	132
7.3.3	Nonlinear equilibrium	133
7.4	Current-Hole	134
7.4.1	Time scheme	135
7.4.2	Variational formulation	135
7.4.3	Numerical results	137
7.5	Conclusions	144

7.1 Introduction

In this chapter, we are interested in some problems that arise in *MHD* models. We begin by studying the case of anisotropic diffusion. It is very important to derive accurate methods for this kind of problems. Usually, physicists use curvilinear meshes. We will show that, at least in this case, the use of higher order methods gives an interesting way to solve this problem without using curvilinear elements. In the second part of this chapter, we treat the MHD equilibrium problem. We will give an overview of different Magneto-static equilibriums, and show that the use of IGA seems to be a very interesting approach. Although the real equilibriums are free boundary problems, in this chapter we will only treat the case of Soloviev solutions. We close our study by an example of instability. For a long time, Plasma physics has been called the science of instabilities. The reason behind that, as explained in [29], is that Plasmas generated in laboratories develop rapid dynamics with a rapid loss of plasma energy which will tend to terminate the plasma discharge. These instabilities are very dangerous because they usually involve large-scale motions and short time scales [29]. The understanding and control of the instabilities is one of the ITER-building challenges.

The second section of this work is inspired from [27], where the authors used cubic Bézier elements. The IGA approach is a natural generalization of Bézier elements. Thanks to IGA, we can reduce considerably the number of degrees of freedom, which is a crucial problem in *JOEKE* code. Moreover, as we have seen with Maxwell's equations, the use of higher regularity elements will allow us to have better CFL condition.

7.2 Anisotropic Diffusion

7.2.1 Introduction

Anisotropic Diffusion plays an important role in tokamak. Several articles [44, 101] have treated this problem, but not in a general complex geometry. The general Anisotropic Diffusion problem writes :

$$\partial_t u - \nabla \cdot (K \nabla u) = f, \quad \Omega \tag{7.2.1}$$

$$u = 0, \quad \partial\Omega \tag{7.2.2}$$

In general, u denotes the temperature (inside the plasma), and K the conductivity.

An important question in Physics, is the choice of the meshes to treat this kind of problems, where we have a strong anisotropy in a given direction. It has been noticed that using methods of lower order, the numerical solution diffuses when crossing edges of elements that not follow the anisotropy. This is why several authors proposed the use of curvilinear meshes so that we can be aligned to the anisotropy. In this section, we show that using methods of higher order, we recover a good behavior of the numerical solution, even if the meshes are not curvilinear and does not follow the anisotropy.

7.2.2 The choice of the grid

In figure 7.1, we give an example of non curvilinear meshes used for our test.

Remark 7.2.1 *In the sequel, we will give results using a cartesian grid to solve the problem. We have tested different meshes and mappings, and we obtained very similar results.*

7.2. Anisotropic Diffusion

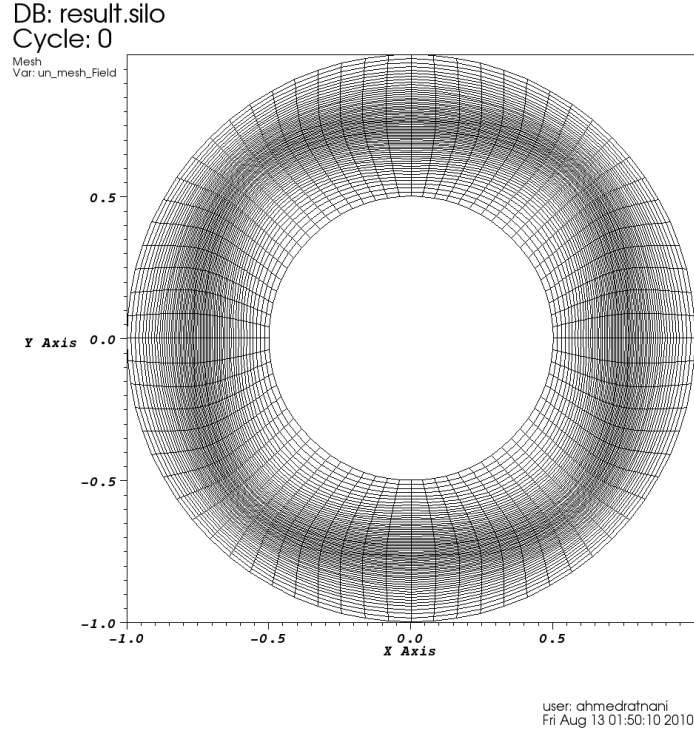


Figure 7.1: Non curvilinear meshes

7.2.3 Evolution of a Gaussian pulse

In this section, we consider the case where :

$$K = \begin{pmatrix} D\mu_{\parallel} \sin^2(\theta) + \mu_{\perp} \cos^2(\theta) & D(\mu_{\parallel} - \mu_{\perp}) \sin(\theta) \cos(\theta) \\ D(\mu_{\parallel} - \mu_{\perp}) \sin(\theta) \cos(\theta) & D\mu_{\parallel} \cos^2(\theta) + \mu_{\perp} \sin^2(\theta) \end{pmatrix} \quad (7.2.3)$$

where μ_{\parallel} , μ_{\perp} and D are constants.

The variational formulation leads to:

$$\partial_t \int_{\Omega} u\varphi - \int_{\Omega} \nabla \cdot (K\nabla u)\varphi = \int_{\Omega} f\varphi, \quad \forall \varphi \in \mathcal{V}_h^0 \quad (7.2.4)$$

we consider here Homogenous Dirichlet boundary condition, as the pulse is localized inside the domain.

By Green's formula we get,

$$\partial_t \int_{\Omega} u\varphi + \int_{\Omega} (K\nabla u) \cdot \nabla \varphi = \int_{\Omega} f\varphi, \quad \forall \varphi \in \mathcal{V}_h^0 \quad (7.2.5)$$

let us expand u_h over the basis of the finite dimension space $\mathcal{V}_h^0 = \mathbf{span}\{\phi_b, b \in \Lambda^0\} \subset H_0^1(\Omega)$,

$$u_h = \sum_{b \in \Lambda} [u]^b \phi_b \quad (7.2.6)$$

7.2. Anisotropic Diffusion

then after discretization, we get,

$$\partial_t \sum_{b \in \Lambda} [u]^b \int_{\Omega} \phi_b \phi_{b'} + \sum_{b \in \Lambda} [u]^b \int_{\Omega} (K \nabla \phi_b) \cdot \nabla \phi_{b'} = \int_{\Omega} f \phi_{b'}, \quad \forall b' \in \Lambda^0 \quad (7.2.7)$$

This is equivalent to the linear system

$$\partial_t M[u] + S[u] = F \quad (7.2.8)$$

where M, S are the Mass and the Stiffness matrices.

We used an implicit time scheme :

$$M[u]^{n+1} = M[u]^n - \Delta t S[u]^{n+1} + F \quad (7.2.9)$$

where for the initialization we took :

$$u(t = 0, x, y) = \psi(x, m_1, \sigma_1) \psi(y, m_2, \sigma_2), \quad \text{with } \psi(x, m, \sigma) = e^{-\frac{(x-m)^2}{2\sigma^2}} \\ f = 0$$

For simulations, we have taken $\sigma_1^2 = \sigma_2^2 = 0.001$, $m_1 = 0.25$, $m_2 = 0.5$, $\mu_{\parallel} = 2.0$, $\mu_{\perp} = 0.0$ and $D = 0.00775$.

In figures 7.2 and 7.3, we show the evolution of the pulse, using a cartesian grid.

Remark 7.2.2 *As we know the behavior of the evolution of the pulse, we can restrict our domain to reduce the dimension of our discrete space. In the case of complex geometry, the IGA approach allows us to solve the problem.*

7.2.4 Conclusions

In plasma physics, we usually use coordinates aligned to flux surfaces, to solve partial differential equations. Indeed, in this case the partial differential equation is reduced to a couple of ordinary differential equations.

The use of higher order elements allows us to treat strong anisotropy, however it is insufficient. The reason is this strong magnetic field in Plasma Confinement. In fact, particles will rotate around the magnetic field lines, while the guiding-center will follow them. In this case, having meshes that follow, or behave like the magnetic field line geometry, is very important, to capture these small variations caused by particles gyration.

7.2. Anisotropic Diffusion

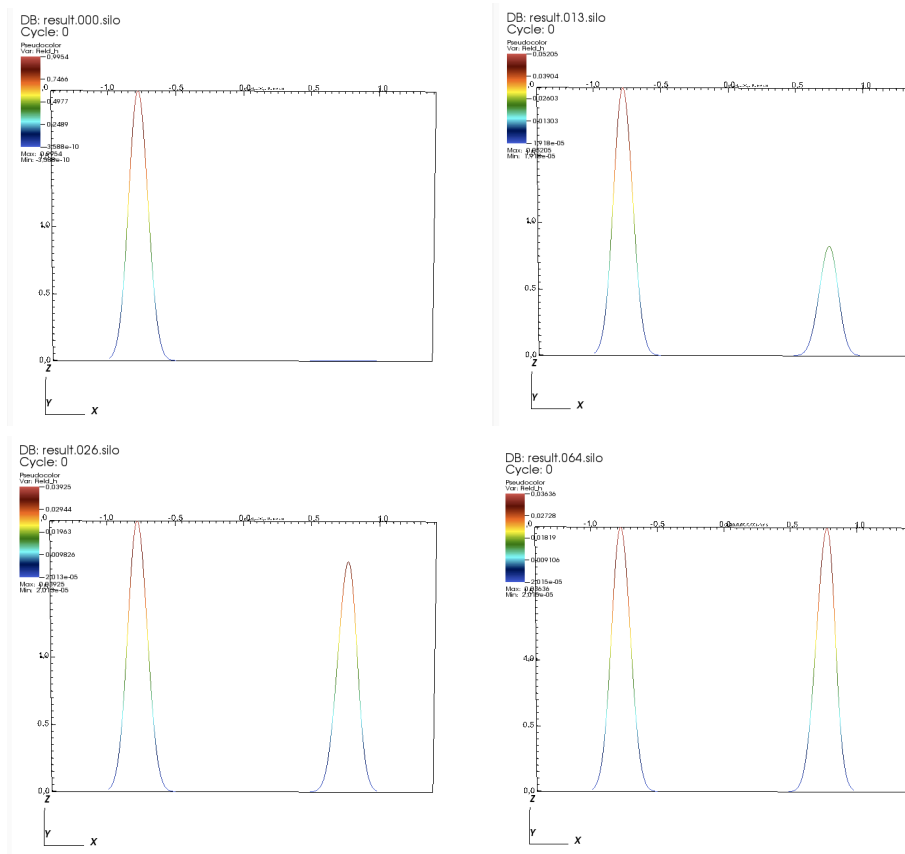


Figure 7.2: Evolution of the pulse, for a radial section, on a square domain

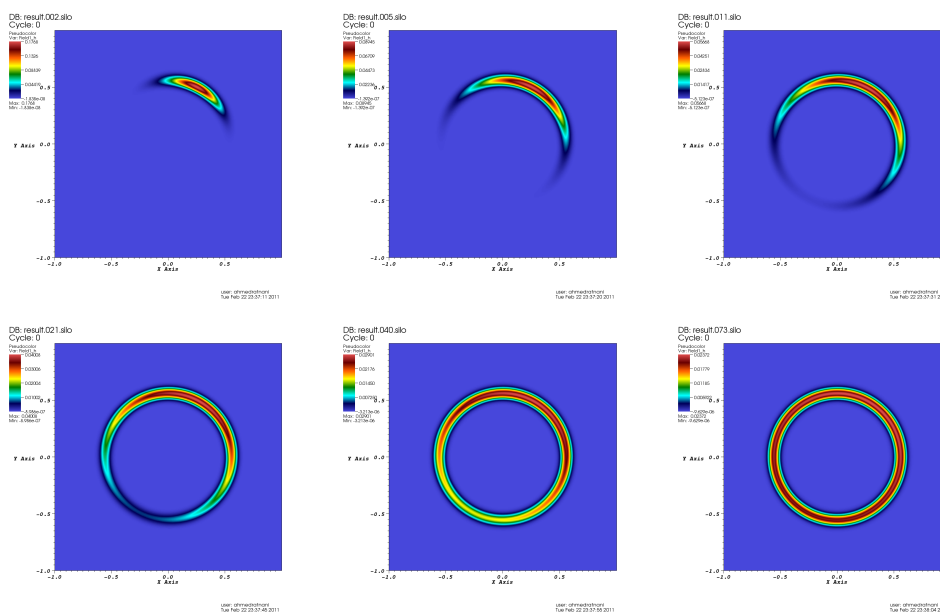


Figure 7.3: Evolution of the pulse, for a radial section, on a square domain

7.3 MHD equilibrium

In MHD, the helical symmetry implies that any physical quantity will depend only on r and $u = l\phi + kz$ (more details can be found in [29]). The general solution of Gauss's law for magnetism $\text{div} \cdot \mathbf{B} = 0$ in helical symmetry can be written in the form:

$$\mathbf{B} = \mathbf{h} \times \nabla\psi(r, u) + \mathbf{h}f(r, u). \quad (7.3.10)$$

where:

- \mathbf{h} is defined by:

$$\mathbf{h} = \frac{r}{l^2 + k^2r^2} \nabla r \times \nabla u,$$

is tangent to the helix $r = \text{const}, u = \text{const}$.

- ψ is the helical flux function. ψ is essentially the component of the vector potential in the direction of the ignorable coordinate [29],

$$\psi = -\frac{\mathbf{A} \cdot \mathbf{h}}{\|\mathbf{h}\|^2} = -(lA_z - krA_\phi)$$

- f is the helical magnetic field. We have:

$$f = \frac{\mathbf{B} \cdot \mathbf{h}}{\|\mathbf{h}\|^2} = (lB_z - krB_\phi)$$

The starting point for the magnetostatic equilibrium (using primitive variables) is the following balance equations (c.f MHD's equations in [71]):

$$\rho(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v}) = -\nabla p + \mathbf{J} \times \mathbf{B} - \rho \nabla \phi_g \quad (7.3.11)$$

where p is the pressure and ϕ_g is the gravitational potential. Usually, in laboratory, this term is negligible. However, it is very important in astrophysical systems.

In the case of a steady state, $\partial_t \cdot = 0$, we end up with,

$$\rho \mathbf{v} \cdot \nabla \mathbf{v} = -\nabla p + \mathbf{J} \times \mathbf{B} \quad (7.3.12)$$

which traduces the balance of forces inside the electrical fluid.

7.3.1 Equilibrium in the absence of toroidal flux

When we can neglect the term $\rho \mathbf{v} \cdot \nabla \mathbf{v}$, the equilibrium writes simply,

$$\nabla p = \mathbf{J} \times \mathbf{B} \quad (7.3.13)$$

The general form of the MHD equilibrium, in this case, is

$$\mathcal{L}_{k,l}\psi = \frac{2kl}{l^2 + k^2r^2} f - f f' - (l^2 + k^2r^2)p' \quad (7.3.14)$$

where the differential operator $\mathcal{L}_{k,l}$ is :

$$\mathcal{L}_{k,l}\psi = \frac{l^2 + k^2r^2}{r} \left\{ \partial_r \frac{r}{l^2 + k^2r^2} \partial_r \psi + \frac{1}{r} \partial_{uu} \psi \right\} \quad (7.3.15)$$

7.3. MHD equilibrium

Plane case

In this case, we have $l = 1$ and $k = 0$, so that $\mathcal{L}_{0,1} = \Delta$. This leads to the equation:

$$\nabla^2 \psi = r^{-1} \partial_r (r \partial_r \psi) + r^{-2} \partial_{\phi\phi} \psi = -f f' - p' \quad (7.3.16)$$

Axisymmetric case

In this case, we have $l = 0$ and $k = 1$, so that $\mathcal{L}_{1,0} = \Delta^*$. It is one of the most important case in tokamaks. This leads to the Grad-Shafranov equation.

$$\Delta^* \psi = r \partial_r (r^{-1} \partial_r \psi) + \partial_{zz} \psi = -f f' - r^2 p' \quad (7.3.17)$$

Soloviev equilibrium

Soloviev equilibrium is a special case of the Grad-Shafranov equation. This happens when the right hand side is independent of ψ . Thus:

$$p = -|p'| \psi + p_0, \quad \text{and} \quad f^2 = f_0^2 + \frac{2\gamma |p'|}{1 + \alpha^2} \psi \quad (7.3.18)$$

The quantities p_0 and $\frac{f_0}{R_0}$ are the pressure and the toroidal field, on the magnetic axis $r = R_0$, $z = 0$, $\psi = 0$.

In this configuration, the Grad-Shafranov equation writes:

$$\Delta^* \psi = |p'| (r^2 - \frac{\gamma}{1 + \eta^2}) \quad (7.3.19)$$

a solution of such equation is :

$$\psi = \frac{|p'|}{2(1 + \eta^2)} ((r^2 - \gamma) z^2 + \frac{\eta^2}{4} (r^2 - R_0^2)^2) \quad (7.3.20)$$

In the sequel, we shall use the cartesian coordinates (x, y) :

$$r = R_0 + ax = R_0(1 + \epsilon x), \quad \text{and} \quad z = ay \quad (7.3.21)$$

where $\epsilon = \frac{a}{R_0}$.

The Soloviev equilibrium writes :

$$-\nabla \cdot \left(\frac{\nabla \psi}{1 + \epsilon x} \right) = a^2 \alpha R_0^2 (1 + \epsilon x) + \frac{a^2 \beta}{1 + \epsilon x} \quad (7.3.22)$$

this can be solved under homogeneous Dirichlet boundary condition, as there exists a level surface where the solution vanishes. We have introduced the quantities:

$$\beta = -\frac{\lambda}{b^2 \epsilon}, \quad \alpha = \frac{4(a^2 + b^2)\epsilon + a^2(2\lambda - \epsilon^3)}{2R_0^2 \epsilon a^2 b^2}$$

Notice that the points $(\pm 1, 0)$ and $(0, \pm \frac{b}{a})$ are on the level surface $\psi = 0$.

In figure 7.4, we show different solutions depending on the ellipticity parameter ϵ , and the triangularity parameter λ .

As one can remark in figure 7.5, increasing the triangularity parameter λ decreases the accuracy of the approximation for a given number of meshes.

7.3. MHD equilibrium

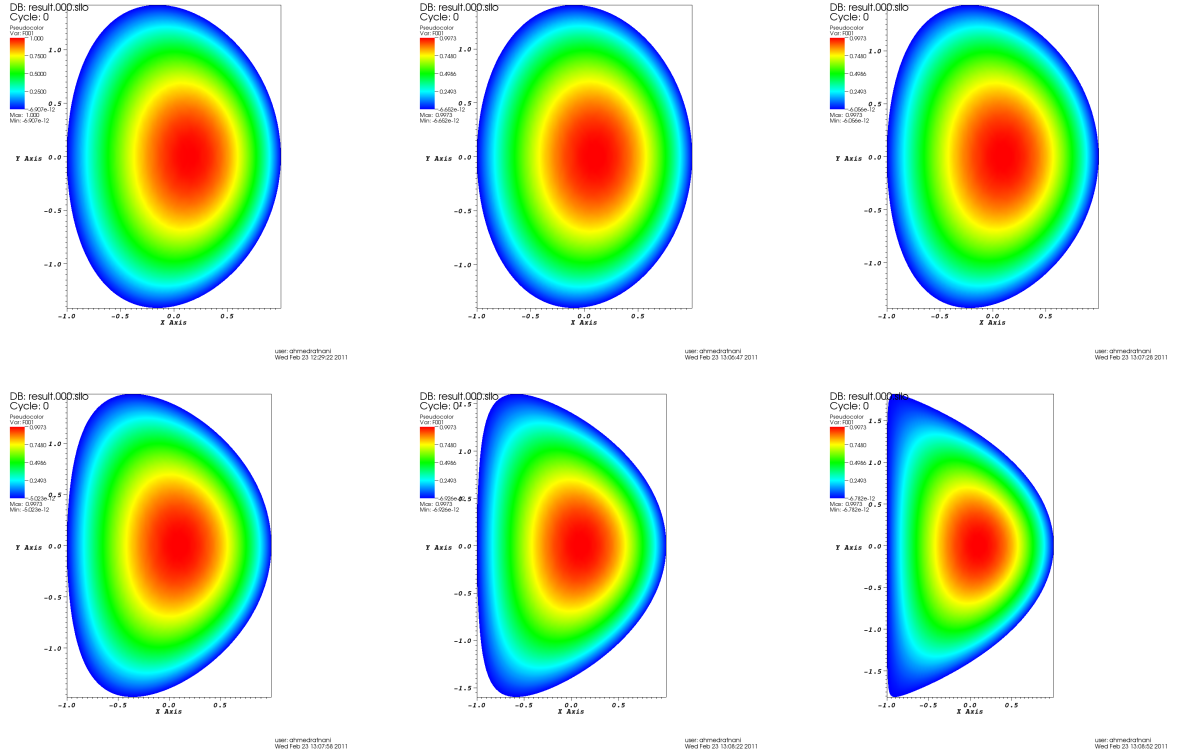


Figure 7.4: 1st line : in order, $(\epsilon, \lambda) = (0.3, 0.0) - test1$, $(0.2, 0.0) - test2$, $(0.2, 0.2) - test3$ 2nd line : in order, $(\epsilon, \lambda) = (0.2, 0.4) - test4$, $(0.2, 0.6) - test5$, $(0.2, 0.7) - test6$

Remark 7.3.1 Using PyIGA, we were able to create meshes that are aligned with the magnetic field lines (c.f figure 7.6).

Remark 7.3.2 In our numerical tests, the boundary was given analytically thanks to Soloviev solutions. In the general case, we need to solve a free boundary problem, (c.f [50, 71] for the physical part, and [112] for the use of IGA in a free boundary problem).

Remark 7.3.3 For the moment, only the construction using linear splines is done with PyIGA. The case of higher degree has been implemented but does not work well because of some memory problems of FITPACK.

7.3.2 Equilibrium with toroidal flux

In this case we keep the term $\rho \mathbf{v} \cdot \nabla \mathbf{v}$, we have,

$$\rho \mathbf{v} \cdot \nabla \mathbf{v} + \nabla p = \mathbf{J} \times \mathbf{B} \quad (7.3.23)$$

we get a generalized Grad-Shafranov equation:

$$\Delta^* \psi = r \partial_r (r^{-1} \partial_r \psi) + \partial_{zz} \psi = -f f' - r^2 \partial_{\psi} p \quad (7.3.24)$$

f is still a function of the only variable ψ , but now the pressure p is a function of r and ψ .

7.3. MHD equilibrium

λ	Relative L^2 error
0.0	4.32E-003
0.2	1.18E-002
0.4	2.35E-002
0.6	3.87E-002
0.7	4.85E-002

Figure 7.5: The evolution of the error depending on the triangulation parameter λ , with $\epsilon = 0.2$, $N = 32$ and $p = 1$

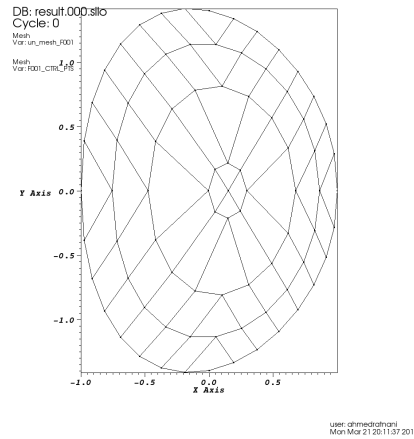


Figure 7.6: Soloviev solution, example of aligned meshes

7.3.3 Nonlinear equilibrium

In this case, we look for the solution of the Grad-Shafranov nonlinear equation :

$$\Delta^* \psi = F(r, \psi) \quad (7.3.25)$$

where, F is a nonlinear function of ψ :

$$F(r, \psi) := -(r^2 f_1(\psi) + f_2(\psi)) \quad (7.3.26)$$

This non linear partial differential equation can be solved using Picard or Newton methods, as described in the chapter 2.

7.4 Current-Hole

The general Current-Hole problem [27] writes:

$$\begin{cases} \partial_t \psi = (1 + \epsilon x)[\psi, \phi] + \eta(J - J_c), \\ \partial_t \omega = 2\epsilon \frac{\partial \phi}{\partial y} \omega + (1 + \epsilon x)[\omega, \phi] + \frac{1}{1 + \epsilon x}[\psi, J] + \nu \Delta \omega, \\ J = \Delta^* \psi, \\ \Delta_{\perp} \phi = \omega. \end{cases}$$

where $[a, b] = \frac{\partial a}{\partial x_1} \frac{\partial b}{\partial x_2} - \frac{\partial a}{\partial x_2} \frac{\partial b}{\partial x_1}$, denotes the Poisson Bracket of the functions a, b .

In the sequel, we consider only the planar cylindrical geometry ($\epsilon = 0$). The problem of the Current-Hole writes :

$$\begin{cases} \partial_t \psi = [\psi, \phi] + \eta(J - J_c), \\ \partial_t \omega = [\omega, \phi] + [\psi, J] + \nu \Delta \omega, \\ J = \Delta \psi, \\ \Delta \phi = \omega. \end{cases} \quad (7.4.27)$$

Remark 7.4.1 In 2D, we can write the Poisson Bracket in a different form. In fact, we have $[a, b] = \nabla a \cdot \text{rot } b = -\text{rot } a \cdot \nabla b$.

The current-hole problem, is subject to the initial conditions:

$$\begin{cases} J(0, \mathbf{x}) = J_c(\mathbf{x}), & \mathbf{x} \in \Omega \\ \psi(0, \mathbf{x}) = \Delta^{-1} J_c, & \mathbf{x} \in \Omega \\ \phi(0, \mathbf{x}) = 0, & \mathbf{x} \in \Omega \\ \omega(0, \mathbf{x}) = 0, & \mathbf{x} \in \Omega \end{cases}$$

and the boundary conditions:

$$\begin{cases} J(t, \mathbf{x}) = 0, & \mathbf{x} \in \partial\Omega, t \in [0, T] \\ \psi(t, \mathbf{x}) = 0, & \mathbf{x} \in \partial\Omega, t \in [0, T] \\ \phi(t, \mathbf{x}) = 0, & \mathbf{x} \in \partial\Omega, t \in [0, T] \\ \omega(t, \mathbf{x}) = 0, & \mathbf{x} \in \partial\Omega, t \in [0, T] \end{cases}$$

Before going through numerical simulations, let us recall some properties for the current-hole problem. We refer to [32, 36] for proofs.

Proposition 7.4.2 (Conservation) *If $\nu = \eta = 0$ then all regular solutions of 7.4.27, verify:*

- Energy conservation :

$$\frac{d}{dt} \int_{\Omega} |\nabla \psi|^2 + |\nabla \phi|^2 d\Omega = 0,$$

- Magnetic Helicity conservation :

$$\frac{d}{dt} \int_{\Omega} \psi d\Omega = 0,$$

- Cross-Helicity conservation :

$$\frac{d}{dt} \int_{\Omega} \nabla \psi \cdot \nabla \phi d\Omega = -\frac{d}{dt} \int_{\Omega} \psi \phi d\Omega = 0.$$

7.4. Current-Hole

As mentioned in [32], when $J_c = 0$ and $\nu > 0$ and $\eta > 0$, we have :

$$\frac{d}{dt} \int_{\Omega} |\nabla \psi|^2 + |\nabla \phi|^2 d\Omega \leq 0.$$

In the sequel, we present the time scheme and variational formulation that we used.

7.4.1 Time scheme

We use a semi-implicit time scheme:

$$\frac{\omega^{n+1} - \omega^n}{\Delta t} = [\omega^n, \phi^n] + [\psi^n, J^n] + \nu \nabla^2 \omega^{n+1}, \quad (7.4.28)$$

$$\nabla^2 \phi^{n+1} = \omega^{n+1}, \quad (7.4.29)$$

$$\frac{\psi^{n+1} - \psi^n}{\Delta t} = [\psi^n, \phi^{n+1}] + \eta \nabla^2 \psi^{n+1} - \eta J_c, \quad (7.4.30)$$

$$J^{n+1} = \nabla^2 \psi^{n+1}. \quad (7.4.31)$$

This time scheme has the advantage of being very simple, however it is of order 1.

7.4.2 Variational formulation

Now let us introduce the discrete space,

$$\mathcal{V}_h^0 = \text{span}\{\varphi_b, \quad b \in \Lambda^0\}.$$

where, the functions φ_b can be *B-splines* or more generally *NURBS*.

Multiplying the equation 7.4.28 by φ_b and taking the integral over the whole domain, we get:

$$\frac{\int_{\Omega} \omega^{n+1} \varphi_b - \int_{\Omega} \omega^n \varphi_b}{\Delta t} = \int_{\Omega} [\omega^n, \phi^n] \varphi_b + \int_{\Omega} [\psi^n, J^n] \varphi_b + \nu \int_{\Omega} \nabla^2 \omega^{n+1} \varphi_b$$

Now using Green's Formulae, we get:

$$\frac{\int_{\Omega} \omega^{n+1} \varphi_b - \int_{\Omega} \omega^n \varphi_b}{\Delta t} = \int_{\Omega} [\omega^n, \phi^n] \varphi_b + \int_{\Omega} [\psi^n, J^n] \varphi_b - \nu \int_{\Omega} \nabla \omega^{n+1} \cdot \nabla \varphi_b$$

thus,

$$\int_{\Omega} \omega^{n+1} \varphi_b + \nu \Delta t \int_{\Omega} \nabla \omega^{n+1} \cdot \nabla \varphi_b = \int_{\Omega} \omega^n \varphi_b + \Delta t \int_{\Omega} [\omega^n, \phi^n] \varphi_b + \Delta t \int_{\Omega} [\psi^n, J^n] \varphi_b$$

Let us consider $\omega_h \in \mathcal{V}_h^0$ an approximation of ω . We can expand ω_h over the basis of \mathcal{V}_h^0 :

$$\omega_h = \sum_{b' \in \Lambda^0} [\omega]^{b'} \varphi_{b'}$$

Therefore, the previous equation leads to the linear to system:

$$A_{\nu}^0[\omega^{n+1}] = M^0[\omega^n] + \Delta t \mathcal{C}_{\mathcal{V}_h^0}[J^n, \psi^n] + \Delta t \mathcal{C}_{\mathcal{V}_h^0}[\phi^n, \omega^n] \quad (7.4.32)$$

7.4. Current-Hole

where we have introduced the matrices,

$$M^0 = \left(\int_{\Omega} \varphi_b \varphi_{b'} \right)_{b,b' \in \Lambda^0}, \quad (7.4.33)$$

$$S^0 = \left(\int_{\Omega} \nabla \varphi_b \cdot \nabla \varphi_{b'} \right)_{b,b' \in \Lambda^0}, \quad (7.4.34)$$

$$A_{\nu}^0 = M^0 + \Delta t \nu S^0 \quad (7.4.35)$$

and $\mathcal{C}_{\mathcal{V}_h^0}[u, v]$ is the L^2 contribution over \mathcal{V}_h^0 of the Poisson's Bracket $[u, v]$, i.e. a column vector where the element of each line b is $\int_{\Omega} [u, v] \varphi_b$.

For the equation 7.4.29, we have for any $\varphi_b \in \mathcal{V}_h^0$:

$$\int_{\Omega} \nabla \phi^{n+1} \cdot \nabla \varphi_b = - \int_{\Omega} \omega^{n+1} \varphi_b$$

which leads to the linear system:

$$S^0[\phi^{n+1}] = -M^0[\omega^{n+1}]. \quad (7.4.36)$$

Now, let's go back to the equation 7.4.30, we have for any $\varphi_b \in \mathcal{V}_h^0$:

$$\frac{\int_{\Omega} \psi^{n+1} \varphi_b - \int_{\Omega} \psi^n \varphi_b}{\Delta t} = \int_{\Omega} [\psi^n, \phi^{n+1}] \varphi_b + \eta \int_{\Omega} \nabla^2 \psi^{n+1} \varphi_b - \eta \int_{\Omega} J_c \varphi_b$$

using Green's Formulae we get:

$$\frac{\int_{\Omega} \psi^{n+1} \varphi_b - \int_{\Omega} \psi^n \varphi_b}{\Delta t} = \int_{\Omega} [\psi^n, \phi^{n+1}] \varphi_b - \eta \int_{\Omega} \nabla \psi^{n+1} \cdot \nabla \varphi_b - \eta \int_{\Omega} J_c \varphi_b.$$

Let us consider $\psi_h \in \mathcal{V}_h^0$ an approximation of ψ . We can expand ψ_h over the basis of \mathcal{V}_h^0 :

$$\psi_h = \sum_{b' \in \Lambda^0} [\psi]^{b'} \varphi_{b'}$$

which leads to:

$$A_{\eta}^0[\psi^{n+1}] = M^0[\psi^n] + \Delta t \mathcal{C}_{\mathcal{V}_h^0}[\psi^n, \phi^{n+1}] - \eta \Delta t M^0[J_c] \quad (7.4.37)$$

with,

$$A_{\eta}^0 = M^0 + \Delta t \eta S^0. \quad (7.4.38)$$

Finally, equation 7.4.31 leads to:

$$M^0[J^{n+1}] = -S^0[\psi^{n+1}]. \quad (7.4.39)$$

Finally, the discretization of the incompressible MHD writes:

$$A_{\nu}^0[\omega^{n+1}] = M^0[\omega^n] + \Delta t \mathcal{C}_{\mathcal{V}_h^0}[J^n, \psi^n] + \Delta t \mathcal{C}_{\mathcal{V}_h^0}[\phi^n, \omega^n], \quad (7.4.40)$$

$$S^0[\phi^{n+1}] = -M^0[\omega^{n+1}], \quad (7.4.41)$$

$$A_{\eta}^0[\psi^{n+1}] = M^0[\psi^n] + \Delta t \mathcal{C}_{\mathcal{V}_h^0}[\psi^n, \phi^{n+1}] - \eta \Delta t M^0[J_c], \quad (7.4.42)$$

$$M^0[J^{n+1}] = -S^0[\psi^{n+1}]. \quad (7.4.43)$$

7.4. Current-Hole

7.4.3 Numerical results

Following [27, 32], we consider the case of a circular domain of radius 1 centered at 0. For initialization, we take :

$$J_c = j_1(1 - R^4) - j_2(1 - R^2)^8.$$

with $R^2 = x^2 + y^2$, $j_1 = 0.2$ and $j_2 = 0.266$. As we can notice in figure 7.7, there is a negative current density close to the axis named the *Current Hole*. In order to start our simulations, we must take the Grad-Shafranov equilibrium based on this current as the initial condition. The following results were taken for $\nu = 10^{-6}$ and $\eta = 10^{-5}$, using

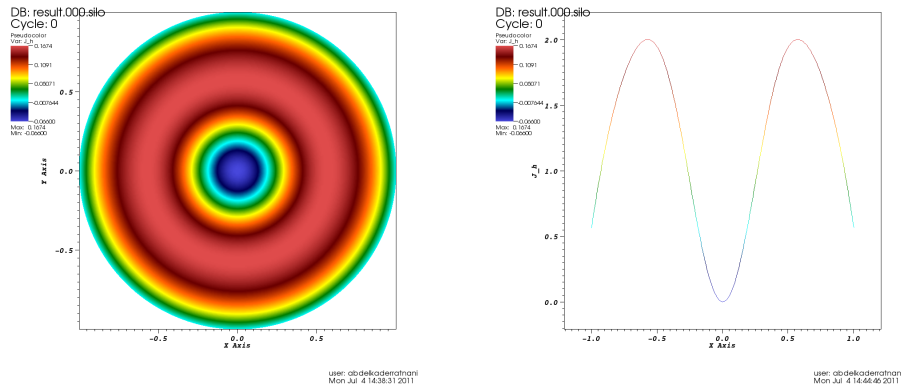


Figure 7.7: *Current Hole*: The current density (left) and its profile (right).

quadratic *NURBS* and a grid of 64×64 meshes. The time step was $dt = 0.1$. In figures 7.8 and 7.9, we plot the evolution of the density current and its profile; they are comparable to the ones obtained in [27, 32]. The current density, initially axisymmetric, is not affected during the linear stage while non-linear effects are still negligible. Then, the profile begin to change: the current density is expelled outward from the central axis, generating a current sheet at the resonant surface. Then, the profile close to the centre is flattened, leaving only residual fluctuations around $J = 0$.

7.4. Current-Hole

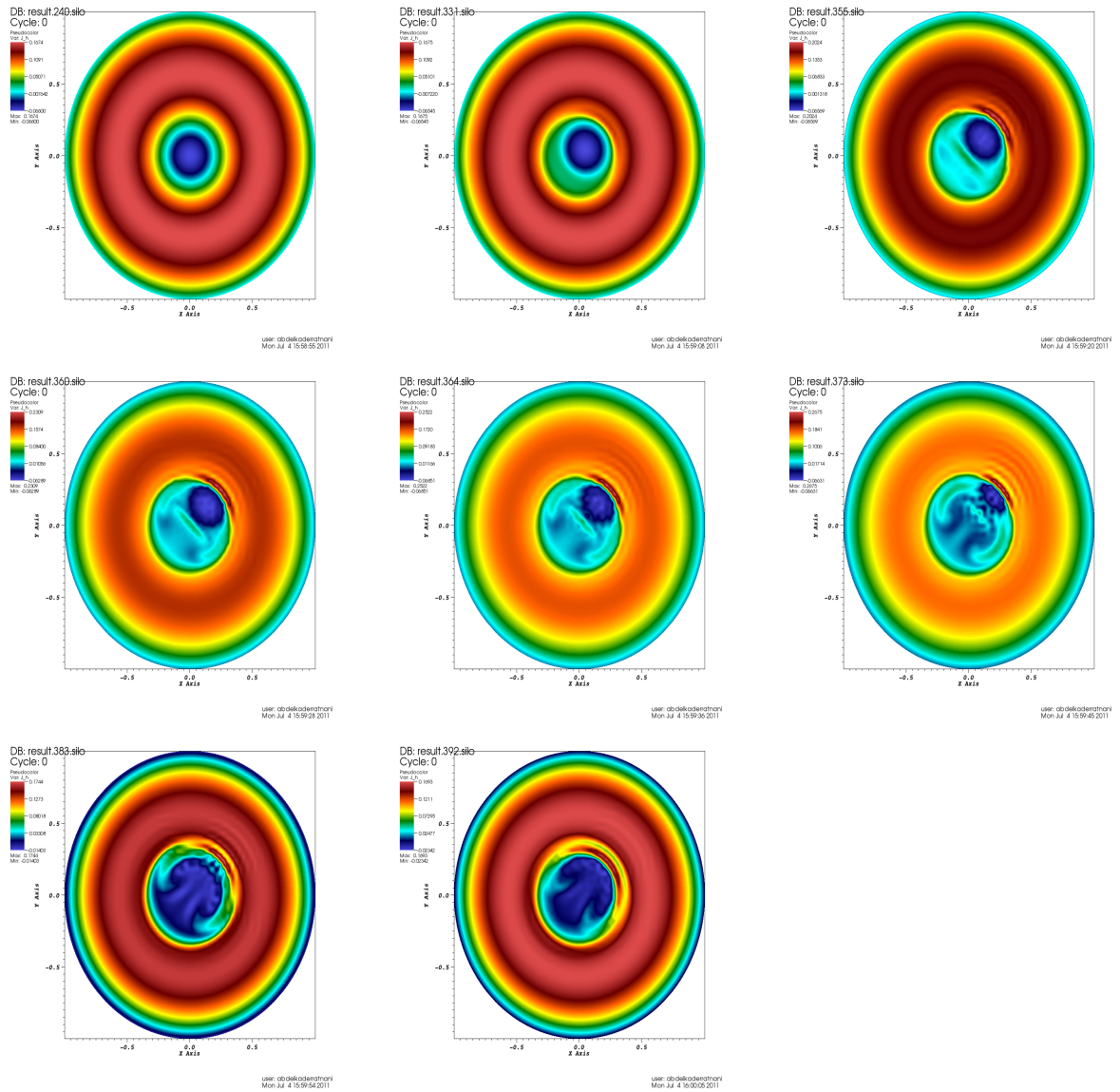


Figure 7.8: *Current Hole:* Evolution of the current density as a function of time for $t = 2400, 3310, 3540, 3600, 3640, 3730, 3830, 3920$, using $\nu = 10^{-6}$ and $\eta = 10^{-5}$.

7.4. Current-Hole

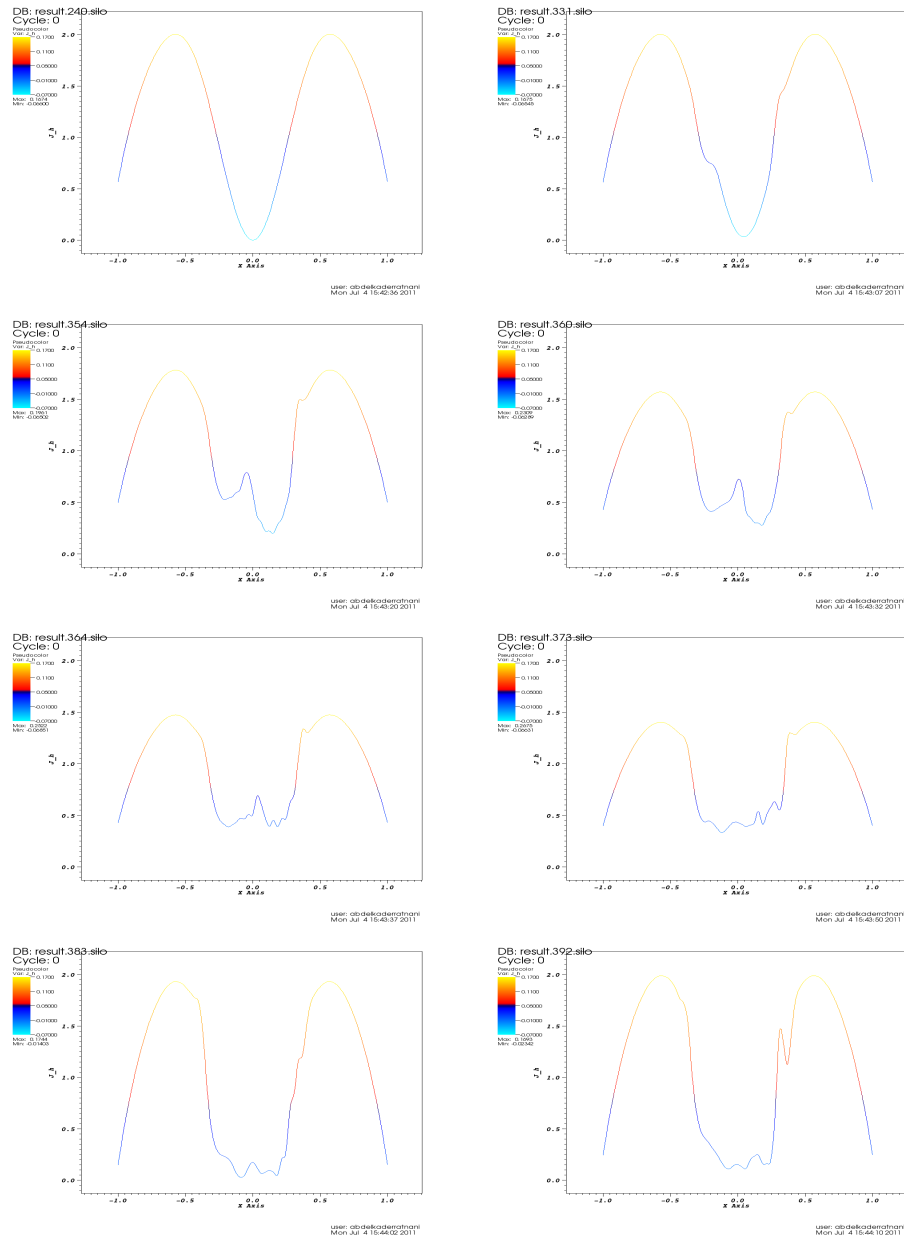


Figure 7.9: *Current Hole:* Evolution of the current density profile as a function of time for $t = 2400, 3310, 3540, 3600, 3640, 3730, 3830, 3920$, using $\nu = 10^{-6}$ and $\eta = 10^{-5}$.

7.4. Current-Hole

In figure 7.10, we plot the kinetic energy as a function of time, for $\nu = 10^{-5}$ and $\eta = 10^{-6}$. In figure 7.11, we show the magnetic, kinetic and total energies as functions of time. As one can see, the impact of the kinetic energy is small. In figure 7.12, we show

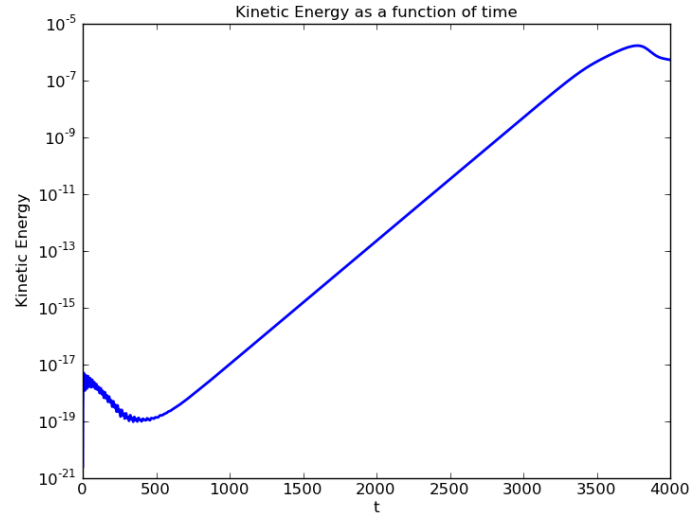


Figure 7.10: *Current Hole:* Kinetic energy of the plasma for $\nu = 10^{-6}$ and $\eta = 10^{-5}$.

the kinetic energy for long times.

7.4. Current-Hole

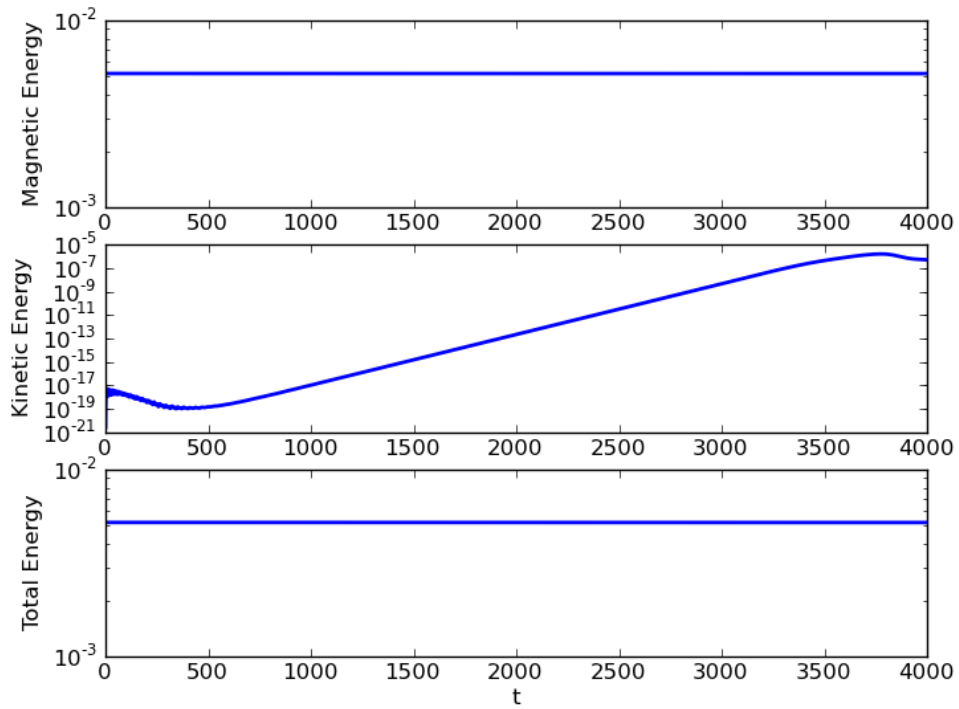


Figure 7.11: *Current Hole:* The energy diagnostic for $\nu = 10^{-6}$ and $\eta = 10^{-5}$.

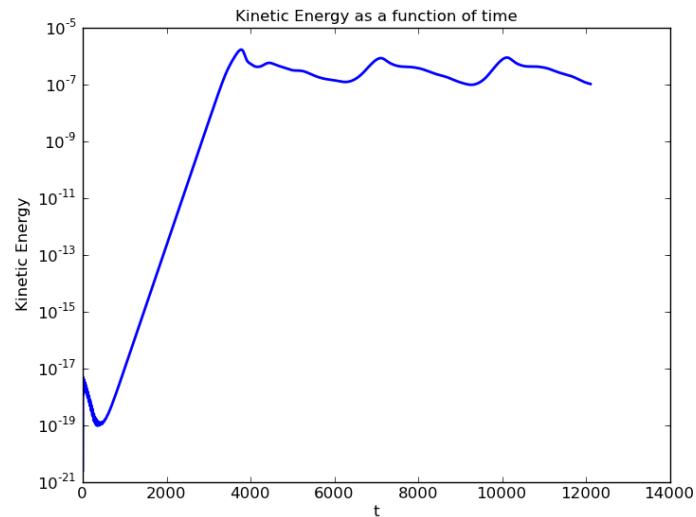


Figure 7.12: *Current Hole:* Kinetic energy of the plasma for $\nu = 10^{-6}$ and $\eta = 10^{-5}$.

7.4. Current-Hole

In order to validate our simulations, we follow [27] and compute the grow rate γ of the $m = 1$ mode during the linear stage. In figure 7.13, we plot the grow rate γ as a function of the resistivity η , for the grids 32×32 and 64×64 using quadratic *NURBS*. It

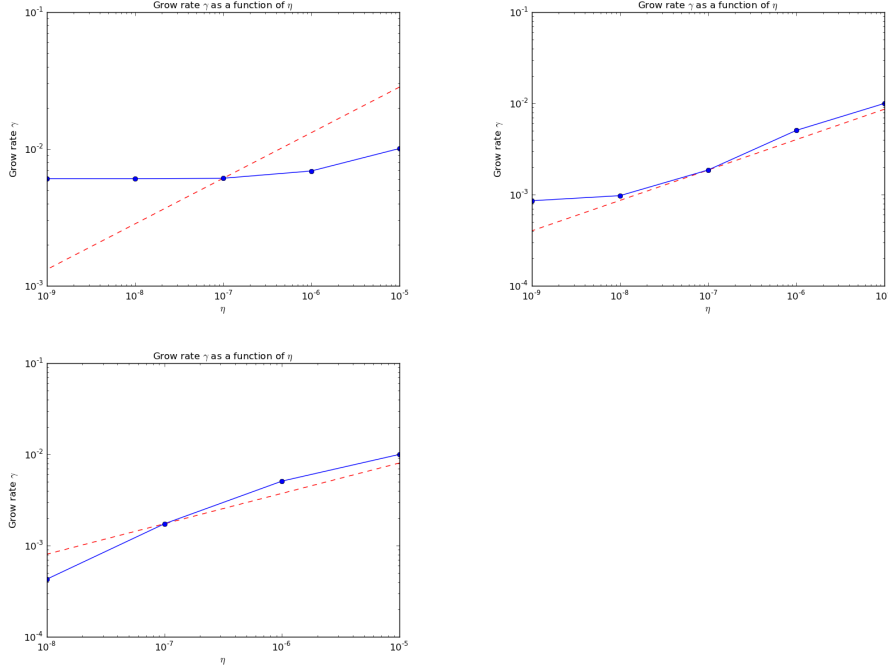


Figure 7.13: *Current Hole*: the grow rate γ as a function of the resistivity η , using quadratic *NURBS*, for the grids : 1^{st} line, (left) 32×32 and (right) 64×64 . 2^{nd} line 128×128 . The dashed line indicates the $\eta^{\frac{1}{3}}$ asymptote.

is very important to notice that our results are not as accurate as those presented in [27]. There many reasons :

- Time scheme : our time scheme is of order 1,
- Local *h-refinement* : In our code *PyIGA*, we do not handle local-refinement.

Remark 7.4.3 (Remark on the instability) *The Soloviev equilibrium, which was the starting point of our simulation, is unstable for the current hole problem. Because of the numerical noise of the simulation, the plasma will leave this equilibrium and start a linear stage and then enter a non-linear one. Using IGA, we can understand exactly the cause of this numerical noise. As we can see in figure 7.14, the singularity of our mapping at the 4 extremities of the patch, will be propagate and force the plasma to leave the equilibrium.*

7.4. Current-Hole

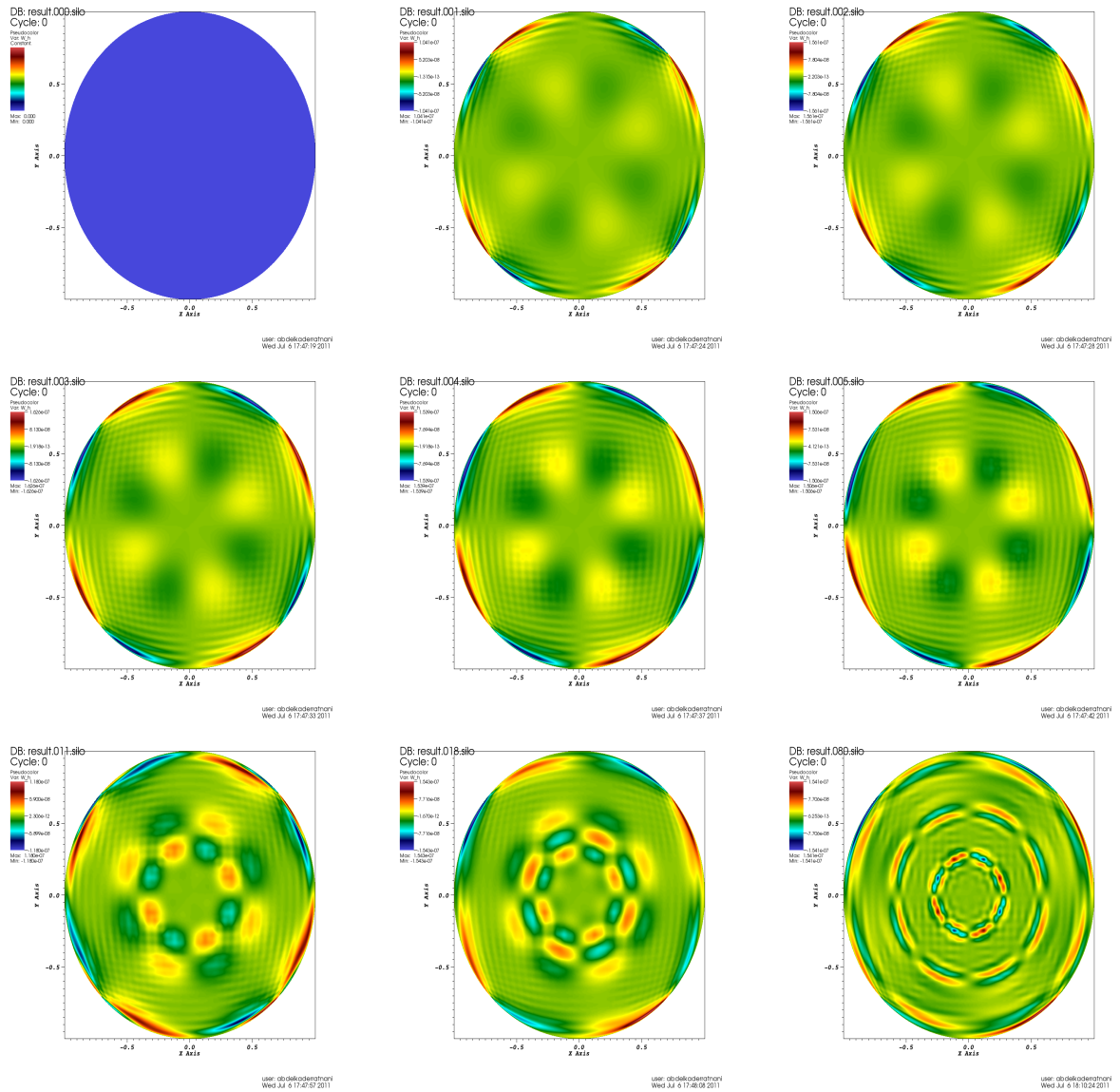


Figure 7.14: *Current Hole:* Creation of the instability. Evolution of ω from left to right and top to bottom at $t = 0, 2, 4, 6, 8, 10, 22, 36, 160$

7.5 Conclusions

In this chapter, we have treated some problems that arise in MHD models. We have seen that the IGA approach, when used in its isoparametric version, gives us a powerful tool to handle realistic geometries that take into account the plasma boundary. Even if we have treated a simple case of equilibrium, we know that IGA provide us necessary tools to solve the free boundary problem. In order to take into account the plasma core, we will need to use local refinement, for example *LR-splines*. Hence, we will reduce considerably the number of degrees of freedom, which is for the moment a crucial problem in *JOEK* [27] code. Another important feature, is the use of the *p-refinement* coupled with a time scheme of order 2, Crank-Nicholson scheme or Adams-Bashforth [32]. As we have noticed it with Maxwell's equations, the use of higher regularity elements gives better CFL numbers. We would like to study these CFL numbers using a second order time scheme. We would like also to use stabilized elements to be able to treat the case where η is very small.

A new DeRham sequence based on Box-splines

Contents

8.1	Introduction	146
8.2	Notations	146
8.3	Bernstein-Bézier bivariate polynomials	147
8.4	Box-Splines	148
8.4.1	Strang-Fix conditions applied to Box-splines	150
8.4.2	Box-Spline series	151
8.4.3	Quasi-interpolant operator for Box-Splines	153
8.5	Box-splines as finite elements basis	153
8.5.1	Approximation with box-splines	154
8.6	DeRham diagram	155
8.6.1	Notations	156
8.6.2	Interpolants and commutativity	156
8.6.3	Approximation Analysis	158
8.7	Boundary Condition using Box-splines	159
8.8	Conclusions and Perspectives	159

8.1 Introduction

We have seen in the chapter 4, the necessity of the construction of DeRham sequences, are very important in modern Numerical Methods in Electromagnetism. In [16, 18], Buffa et al have constructed such sequence using *B-splines*. The problem of such a construction is that it relies on a tensor product. In many problems, we can not use such description. Hence, we need to construct new DeRham sequences based on more sophisticated bivariate splines. In this chapter, we will construct a DeRham sequence using Box-splines, following the same idea as for *B-splines*. The use of Box-splines in order to solve partial differential equation is not new. Hollig et al have used them in a Web-spline method [62]. The first remark about Box-splines is that the support is a union of a finite number of specific triangles. However, the construction of discrete splines spaces on triangles, for a given regularity at edges, is more complicated. Ming-Jan Lai et al have succeeded to use such spaces to solve Stokes and Navier-Stokes equations [78]. However, we can not expect to use such construction for a DeRham sequence. We will begin by a brief recall on Box-splines. All definitions and properties were taken from [99, 30].

For our construction of the DeRham sequence, we will not take into account general boundary conditions, even if it is a crucial question. We shall only consider the periodic case. In the last section, we will show how to construct Box-splines that are interpolating at the boundary, without giving any proof of the approximation accuracy using such elements.

8.2 Notations

Let us define the vectors $\mathbf{e}^1 := (1, 0)$, $\mathbf{e}^2 := (0, 1)$, $\mathbf{e}^3 := (1, 1)$ and $\mathbf{e}^4 := (-1, 1)$.

In the sequel, $X_n := \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\}$ will denote a direction set.

The idea behind box-splines, is that starting with the characteristic function of the unit square, we will use convolution in several directions to generate a new function, called box-spline. We can recover the case of bivariate B-splines if we do convolution using the two directions \mathbf{e}^1 and \mathbf{e}^2 .

A direction set is said to be of type-I, if $\mathbf{v}_i \in \{\mathbf{e}^1, \mathbf{e}^2, \mathbf{e}^3\}$, and $\forall n \geq 3, \mathbf{e}^1, \mathbf{e}^2, \mathbf{e}^3 \in X_n$. It is of type-II, if $\mathbf{v}_i \in \{\mathbf{e}^1, \mathbf{e}^2, \mathbf{e}^3, \mathbf{e}^4\}$, and $\forall n \geq 4, \mathbf{e}^1, \mathbf{e}^2, \mathbf{e}^3, \mathbf{e}^4 \in X_n$.

Without loss of generality, let us consider $\mathbf{v}_i = \mathbf{e}^i, \forall i \in \{1, 2, 3\}$. Let $X_3 = \{\mathbf{v}^1, \mathbf{v}^2, \mathbf{v}^3\}$ and $\forall i \geq 4, X_i := X_{i-1} \cup \{\mathbf{v}_i\}$.

For any function f of real values, let $\delta_{\mathbf{u}}f := f(\cdot) - f(\cdot - \mathbf{u})$ and $\delta^{\mathbf{u}}f := f(\cdot + \mathbf{u}) - f(\cdot)$ the backward and forward difference.

Let Δ be a triangulation. The space of splines, of a regularity r across edges and of degree d , is

$$\mathcal{S}_d^r(\Delta) = \{s \in C^r(\Delta), \quad s|_{T_i} \in \mathcal{P}_d, \quad \forall 1 \leq i \leq |\Delta|\}$$

For simplicity, we enumerate triangles so that $\Delta = \{T_j, 1 \leq j \leq |\Delta|\}$.

We define m_α the monomial of multi-index α :

$$m_\alpha(v) := \frac{1}{\alpha!} x^{\alpha_1} y^{\alpha_2}, \quad \alpha = (\alpha_1, \alpha_2) \in \mathbb{Z}_+^2$$

where for any multi-index, we denote by $\alpha! = \alpha_1! \alpha_2!$

8.3 Bernstein-Bézier bivariate polynomials

Before introducing the Box-splines, we shall define Bernstein polynomials for triangles. We have the following result:

Lemma 8.3.1 $\forall \mathbf{v} \in \mathbb{R}^2, \exists (b_1, b_2, b_3) \in \mathbb{R}^3, \mathbf{v} = b_1 \mathbf{v}_1 + b_2 \mathbf{v}_2 + b_3 \mathbf{v}_3$ such that $b_1 + b_2 + b_3 = 1$

The values of the coefficients b_i are $b_i = \frac{A_{T_i}}{A_T}$ where $T = \langle \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \rangle$ and T_i is obtained by changing \mathbf{v}_i by \mathbf{v} . A_T is the area of the triangle T . Therefore, for $\mathbf{v} = (x, y)$, the b_i are linear polynomials of x and y . The coefficients b_i are said the barycentric coordinates.

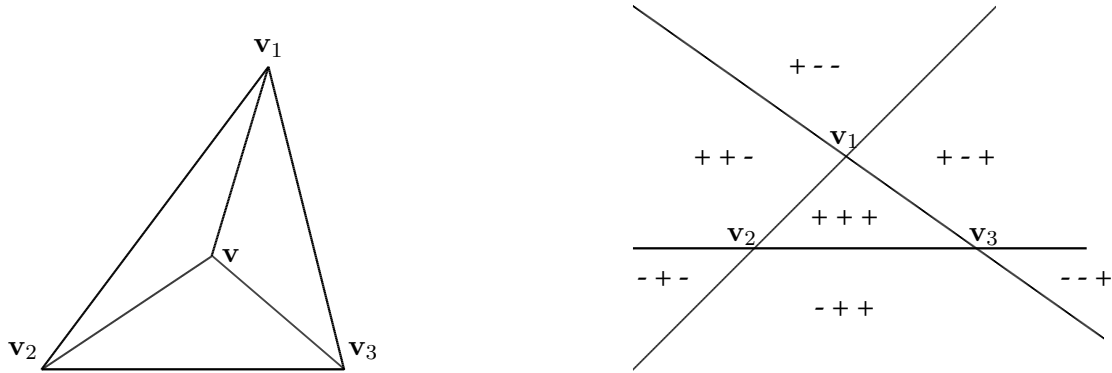


Figure 8.1: (left) a triangle element $T = \langle \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \rangle$, (right) sign of the coefficients b_1, b_2, b_3

Definition 8.3.2 (Bernstein basis polynomials) The Bernstein polynomials of degree d are $B_{ijk}^d(\mathbf{v}) = \frac{d!}{i!j!k!} b_1^i b_2^j b_3^k$, where $i + j + k = d, i, j, k \in \mathbb{N}^*$.

Bernstein polynomials have the property of the partition of unity: $1 = (b_1 + b_2 + b_3)^d = \sum_{i+j+k=d} \frac{d!}{i!j!k!} b_1^i b_2^j b_3^k = \sum_{i+j+k=d} B_{ijk}^d$. We also have $\forall \mathbf{v} \in T, 0 \leq B_{ijk}^d(\mathbf{v}) \leq 1$.

Theorem 8.3.3 Let $\mathcal{B}^d = \{B_{ijk}^d\}_{i+j+k=d}$, \mathcal{B}^d is a basis for the space \mathcal{P}_d of bivariate polynomials. For each $p \in \mathcal{P}_d$ the B-form of p is $p = \sum_{i+j+k=d} c_{ijk} B_{ijk}^d$

Let $\xi_{ijk} = \frac{i\mathbf{v}_1 + j\mathbf{v}_2 + k\mathbf{v}_3}{d}$. The set $\mathcal{D}_{d,T} = \{\xi_{ijk}, i + j + k = d\}$ is the set of domain-points.

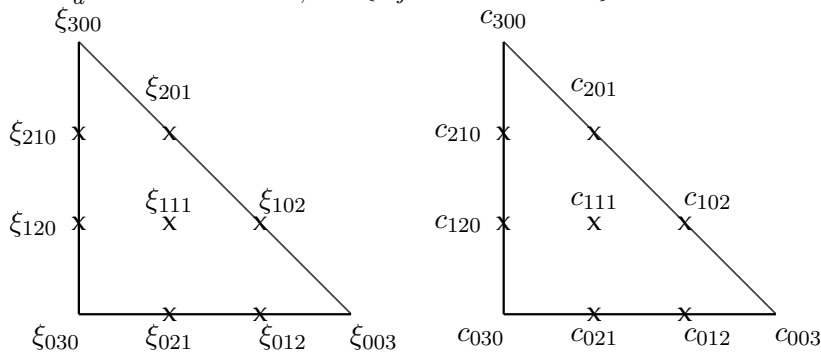


Figure 8.2: (left) Domain points and (right) B-coefficients for a cubic polynomial

As each spline has a unique B-form on each triangle. We can now control the regularity of a spline over a triangulation, thanks to its B-coefficients.

8.4 Box-Splines

This is a brief sketch of the Box-splines functions, and some of their properties.

Definition 8.4.1 (Box-Spline of type I) For $4 \leq i \leq n$, we define the box-spline of type-I associated to the direction set X_n recursively by

$$B(\mathbf{v}|X_i) := \int_0^1 B(\mathbf{v} - t\mathbf{v}_i|X_{i-1})dt$$

where $B(\mathbf{v}|X_3) = B_{111}$ is the hat box-spline (Courant element).

In figure 8.3, we show the support and B-coefficients of the Courant element.

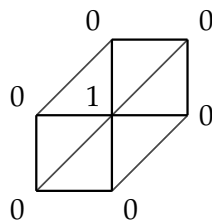


Figure 8.3: The support and B-coefficients of the Box-spline B_{111}

Definition 8.4.2 (Box-Spline of type II) For $5 \leq i \leq n$, we define the box-spline of type-II associated to the direction set X_n recursively by

$$B(\mathbf{v}|X_i) := \int_0^1 B(\mathbf{v} - t\mathbf{v}_i|X_{i-1})dt$$

where $B(\mathbf{v}|X_4) = B_{1111}$ is the ZP element.

In figure 8.4, we show the impact of convolution in the directions \mathbf{e}_3 and \mathbf{e}_4 . In figure

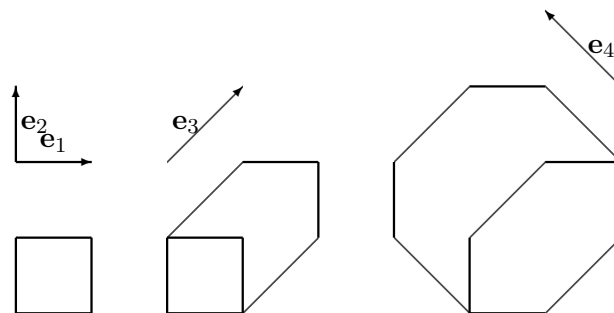


Figure 8.4: Construction of Box-splines: Starting with the characteristic function of the unit square (left), convolution in the direction \mathbf{e}^3 (middle) gives the hat (Courant) function, further convolution in the direction \mathbf{e}^4 (right) gives the ZP element

8.6, we show the supports of the Box-splines B_{211} , B_{221} , B_{222} , and B_{322} . In figure 8.7, we show the B-net (B-coefficients) of $2 \cdot B_{111}$.

8.4. Box-Splines

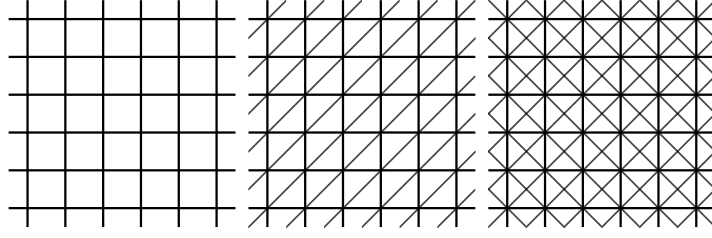


Figure 8.5: The two-, three-, and four-direction meshes

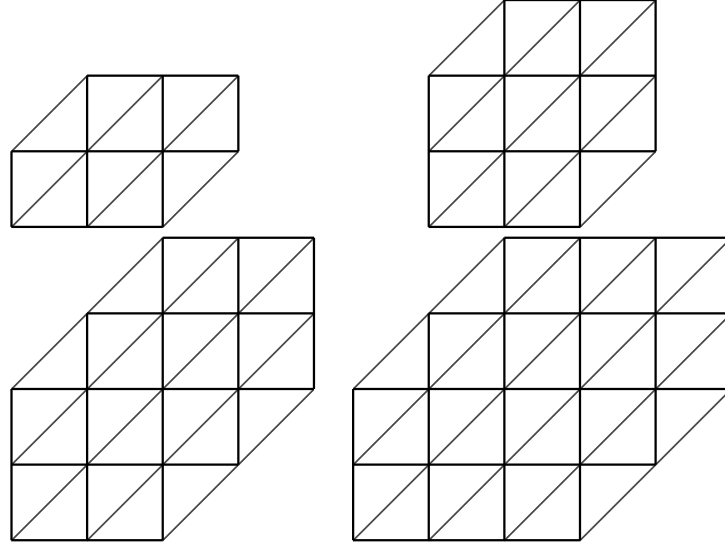


Figure 8.6: The supports of B_{211} , B_{221} , B_{222} , and B_{322} .

Theorem 8.4.3 *The support of $B(\mathbf{v}|X_n)$ is the closure of the set*

$$[X_n] := \left\{ \sum_{j=1}^n t_j \mathbf{v}_j : 0 \leq t_j < 1, j \in \{1, \dots, n\} \right\}$$

Moreover, $B(\mathbf{v}|X_n) > 0$, for all \mathbf{v} in the interior of $[X_n]$

Proposition 8.4.4 *For any $4 \leq j \leq n$, we have*

$$\partial_{\mathbf{v}_j} B(\cdot|X_n) = \delta_{\mathbf{v}_j} B(\cdot|X_n - \{\mathbf{v}_j\})$$

In what follows, we shall denote B as the box-spline associated with the direction set X_n of type-I (or type-II). We will denote also: $B^x(\cdot) := B(\cdot|X_n - \{\mathbf{e}_1\})$, $B^y(\cdot) := B(\cdot|X_n - \{\mathbf{e}_2\})$. We can write $X_n = E_1 \cup E_2 \cup E_3$ (or $X_n = E_1 \cup E_2 \cup E_3 \cup E_4$), with $E_i := \{e^i, \dots, e^i\}$, where $|E_i| = n_i$, and $n_1 + n_2 + n_3 = n = |X_n|$ (or $n_1 + n_2 + n_3 + n_4 = n = |X_n|$).

Theorem 8.4.5 (Regularity of Box-splines) *For Box-splines of type-I, we have $B_{n_1 n_2 n_3} = B(\cdot|X_n) \in \mathcal{S}_{n-2}^r(\Delta_I)$, where $r := r(X_n) = \min\{n_1 + n_2, n_2 + n_3, n_3 + n_1\} - 2$. For Box-splines of type-II, we have $B_{n_1 n_2 n_3 n_4} = B(\cdot|X_n) \in \mathcal{S}_{n-2}^r(\Delta_{II})$, where $r := r(X_n) = \min\{n_1 + n_2 + n_3, n_2 + n_3 + n_4, n_3 + n_4 + n_1, n_4 + n_1 + n_2\} - 2$.*

Theorem 8.4.6 $\forall f \in C(\mathbb{R}^2)$, $\int_{\mathbb{R}^2} B(\mathbf{v}) f(\mathbf{v}) d\mathbf{v} = \int_{[0,1]^n} f(\sum_{i=1}^n t_i \mathbf{v}_i) dt_1 \cdots dt_n$

8.4. Box-Splines

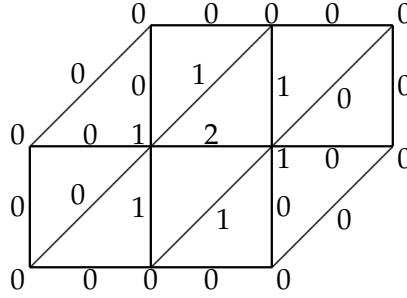


Figure 8.7: The B-net (B-coefficients) of $2 \cdot B_{111}$

Theorem 8.4.7 $\forall n \geq 3, \forall f \in C^1(\mathbb{R}^2), \int_{\mathbb{R}^2} B(\mathbf{v}) \partial_{v_j} f(\mathbf{v}) d\mathbf{v} = - \int_{\mathbb{R}^2} \partial_{v_j} B(\mathbf{v}) f(\mathbf{v}) d\mathbf{v}$

Theorem 8.4.8 $\forall f \in C^n(\mathbb{R}^2), \int_{\mathbb{R}^2} B(\mathbf{v}) \partial_{X_n} f(\mathbf{v}) d\mathbf{v} = \delta^{X_n} f(0,0)$

Theorem 8.4.9 (h-refinement)

$$B(\mathbf{v}|X_n) = \sum_{\nu \in \mathbb{Z}^2} a_\nu B(2\mathbf{v} - \nu|X_n)$$

Theorem 8.4.10 (scalar product) If $n_1 + n_2 + n_3 = m_1 + m_2 + m_3 = n$, and $n_i, m_i > 0$,

$$\int_{\mathbb{R}^2} B_{n_1, n_2, n_3} B_{m_1, m_2, m_3} = B_{n_1+m_1, n_2+m_2, n_3+m_3}(n_1 + n_3, n_2 + n_3)$$

8.4.1 Strang-Fix conditions applied to Box-splines

In this section, we recall an interesting theorem from Strang-Fix [99].

Definition 8.4.11 (Strang-Fix conditions) A compactly supported function $\phi \in C(\mathbb{R}^2)$, such that its Fourier transform $\hat{\phi} \in L^1(\mathbb{R}^2)$, is said to satisfy the **Strang-Fix** conditions with **SF index** $\alpha \in \mathbb{Z}_+^2$ if:

- $\hat{\phi}(0,0) = 1$,
- $D^\gamma \hat{\phi}(2\pi\nu) = 0, \quad \forall \nu \in \mathbb{Z}^2 - \{(0,0)\}, \gamma \leq \alpha$

The collection Λ_ϕ of all SF indices of ϕ is called the **SF indicator set** of ϕ . The largest integer n for which $\alpha \in \Lambda_\phi$ whenever $|\alpha| \leq n$ is called the **SF degree** of ϕ .

We have the following nice result

Theorem 8.4.12 If X_n is of type I or II, then the SF degree of the box-spline $B(\cdot|X_n)$ is $r(X_n)+1$, where $r(X_n)$ is the regularity of $B(\cdot|X_n)$.

Henceforth, we shall denote by Λ_{X_n} the SF indicator set of the box-spline $B(\cdot|X_n)$. When we will omit X_n for the box-spline, its SF indicator set will be Λ , r will be its regularity.

8.4. Box-Splines

8.4.2 Box-Spline series

We define the linear space of all splines :

$$\mathcal{S}(X_n) := \left\{ \sum_{\nu \in \mathbb{Z}^2} a_\nu B(\cdot - \nu | X_n); a_\nu \in \mathbb{R}, \nu \in \mathbb{Z}^2 \right\}$$

For simplicity, when we fixe a direction set X_n , we shall denote by B_ν the box-spline $B(\cdot - \nu | X_n)$, for $\nu \in \mathbb{Z}^2$. $\sigma(B_\nu)$ will be its support. We will denote \mathcal{S} in stead of $\mathcal{S}(X_n)$.

For each triangle T_j , Λ_j will be the set of Box-splines indices, that are none vanishing on T_j , $\Lambda_j = \{\nu \in \mathbb{Z}^2, B_\nu|_{T_j} \neq 0\}$. We denote by \tilde{T}_j its extension, i.e $\tilde{T}_j = \cup_{\nu \in \Lambda_j} \sigma(B_\nu)$.

Theorem 8.4.13 (Global Independence) *The Box-splines $\{B_\nu(\cdot)\}_{\nu \in \mathbb{Z}^2}$ are linearly independent.*

There is also a local version of the last property. Jia [72] have shown :

Theorem 8.4.14 (Local Independence) *For any open set $A \in \mathbb{R}^2$, the Box-splines $\{B_\nu(\cdot), \nu \in \mathbb{Z}^2, \sigma(B_\nu) \cap A \neq \emptyset\}$ are linearly independent.*

Theorem 8.4.15 (Stability) *If $a := \{a_\nu\}_{\nu \in \mathbb{Z}^2}$ is a bounded sequence, K a compact in \mathbb{R}^2 , and $1 \leq p \leq \infty$:*

$$A \|a\|_p |K|^{\frac{1}{p}} \leq \left\| \sum_{\nu \in \mathbb{Z}^2} a_\nu B_\nu \right\|_{p,K} \leq \|a\|_p |K|^{\frac{1}{p}}$$

for some constant A .

Proof We will distinguish the case $1 \leq p < \infty$ from $p = \infty$. We only treat here the case $1 \leq p < \infty$, the other one is similar. First, we begin by the second inequality which is more simple. We have

$$\begin{aligned} \left\| \sum_{\nu \in \mathbb{Z}^2} a_\nu B_\nu \right\|_{p,K}^p &= \int_K \left(\sum_{\nu \in \mathbb{Z}^2} a_\nu B_\nu \right)^p \\ &\leq \int_K \left(\sum_{\nu \in \mathbb{Z}^2} a_\nu^p \right) \left(\sum_{\nu \in \mathbb{Z}^2} B_\nu^p \right) \\ &\leq \int_K \left(\sum_{\nu \in \mathbb{Z}^2} a_\nu^p \right) \left(\sum_{\nu \in \mathbb{Z}^2} B_\nu \right) \\ &\leq \int_K \left(\sum_{\nu \in \mathbb{Z}^2} a_\nu^p \right) = \|a\|_p^p |K| \end{aligned}$$

where we used Holder inequality, the fact that the box-splines are positive and form a partition of unity.

For the first inequality, we assume the contrary. Let $a^m := \{a_\nu^m\}_{\nu \in \mathbb{Z}^2}$, be uniformly bounded sequences such that $\|a^m\|_p |K|^{\frac{1}{p}} = 1$, and

$$\left\| \sum_{\nu \in \mathbb{Z}^2} a_\nu^m B_\nu \right\|_{p,K} \rightarrow 0$$

As the sequence is bounded, we can then extract a sub-sequence that converges to some $a = \{a_\nu\}_{\nu \in \mathbb{Z}^2}$. Then, we have, by passing to the limit

$$\|a\|_p |K|^{\frac{1}{p}} = 1, \quad \left\| \sum_{\nu \in \mathbb{Z}^2} a_\nu B_\nu \right\|_{p,K} = 0$$

8.4. Box-Splines

therefore, $\sum_{\nu \in \mathbb{Z}^2} a_\nu B_\nu = 0$, so that $a = 0$, which contradicts the fact that $\|a\|_p |K|^{\frac{1}{p}} = 1$. \square

Theorem 8.4.16 (Stability) *If $a := \{a_\nu\}_{\nu \in \mathbb{Z}^2}$ is a bounded sequence, K a compact in \mathbb{R}^2 , and $1 \leq p \leq \infty$:*

$$A \|a\|_p |K|^{\frac{1}{p}} \leq \left\| \sum_{\nu \in \mathbb{Z}^2} a_\nu B_\nu \right\|_{p,K} \leq \|a\|_p |K|^{\frac{1}{p}}$$

for some constant A .

Theorem 8.4.17 (Marsden's identity) *Let $L_0(\cdot) = 1$, and for $\beta \in \mathbb{Z}_+^2$ we define L_β by induction by*

$$L_\beta(\cdot) := m_\beta(\cdot) - \sum_{j \in \mathbb{Z}^2} B(j) \sum_{\gamma \leq \beta, \gamma \neq \beta} \frac{(-j)^{\beta-\gamma}}{(\beta-\gamma)!} L_\gamma(\cdot)$$

then we have the Marsden's identity

$$m_\alpha(v) = \sum_{\nu \in \mathbb{Z}^2} L_\alpha(\nu) B_\nu(v), \quad \forall v \in \mathbb{R}^2, \alpha \in \Lambda$$

Theorem 8.4.18 (Polynomial reproducing) *For any multivariate polynomial $p \in \mathcal{P}_r$, there exist a unique real sequence $\{c_\nu\}_{\nu \in \mathbb{Z}^2}$ such that*

$$p(\cdot) = \sum_{\nu \in \mathbb{Z}^2} c_\nu B_\nu(\cdot)$$

Proof We know that $m_\alpha(v) = \sum_{\nu \in \mathbb{Z}^2} L_\alpha(\nu) B_\nu(v)$, $\forall v \in \mathbb{R}^2, \alpha \in \Lambda$. Let $p \in \mathcal{P}_r$. By applying the Taylor expansion to p , we have

$$p(v) = \sum_{\alpha \in \mathbb{Z}^2, |\alpha| \leq r} \frac{\partial_\alpha p(0)}{\alpha!} m_\alpha(v)$$

then by the Marsden's identity we have

$$p(v) = \sum_{\alpha \in \mathbb{Z}^2, |\alpha| \leq r} \frac{\partial_\alpha p(0)}{\alpha!} \sum_{\nu \in \mathbb{Z}^2} L_\alpha(\nu) B_\nu(v)$$

which leads to

$$p(v) = \sum_{\nu \in \mathbb{Z}^2} \left\{ \sum_{\alpha \in \mathbb{Z}^2, |\alpha| \leq r} \frac{\partial_\alpha p(0)}{\alpha!} L_\alpha(\nu) \right\} B_\nu(v)$$

which proves the existence. The unicity, is a consequence of the linear independence of box-spline series (cf th 8.4.13,8.4.14). \square

We now, can define functionals $\{c_\nu\}_{\nu \in \mathbb{Z}^2}$, such that, for each multivariate polynomial $p \in \mathcal{P}_r$, we have $c_\nu(p) := c_\nu$, where c_ν is the coefficient associated to B_ν in the latest expansion. The result is easily extensible to the space of splines, by a domain decomposition. Now we would like to extend the definition of those functionals to L^p spaces.

Lemma 8.4.19 *In the case of $\Omega = \mathbb{R}^2$, the Box-splines series $\sum_{\nu \in \mathbb{Z}^2} c_\nu(s) B_\nu$ converges to s , for each $s \in \mathcal{S} \cap L^p$, $0 < p < \infty$ in the L^p metric.*

8.5. Box-splines as finite elements basis

Proof Let $s \in \mathcal{S} \cap L^p$ then by the stability property of box-splines series we get $\|c(s)\|_p < \infty$. Then, as $m \rightarrow \infty$, we have

$$\|s - \sum_{\nu \in \mathbb{Z}^2, |\nu| \leq m} c_\nu(s) B_\nu\|_p^p = \left\| \sum_{\nu \in \mathbb{Z}^2, |\nu| \not\leq m} c_\nu(s) B_\nu \right\|_p^p \lesssim \sum_{\nu \in \mathbb{Z}^2, |\nu| \not\leq m} c_\nu(s)^p \rightarrow 0$$

□

For $p = 1$ and $\nu \in \mathbb{Z}^2$, we see that c_ν is a bounded linear functional on \mathcal{S} in the L^1 norm. Let λ_ν be its Hahn-Banach extension onto $L^1(T_j)$ we will have $|\lambda_\nu(f)| \lesssim |T_j|^{-1} \|f\|_{1, T_j}$. By Holder inequality we get

$$|\lambda_\nu(f)| \lesssim |T_j|^{-p} \|f\|_{p, T_j}, \quad \forall 1 \leq p \leq \infty$$

8.4.3 Quasi-interpolant operator for Box-Splines

We are now able to define a quasi-interpolant operator for Box-splines series.

Theorem 8.4.20 (Quasi-interpolant) For each locally integrable function $f \in L^1_{loc}(\Omega)$, we define the operator

$$Q(f) := \sum_{\nu \in \mathbb{Z}^2} \lambda_\nu(f) B_\nu$$

The quasi-interpolant Q is a projection from $L^p(\Omega)$, $1 \leq p \leq \infty$ onto its subspace \mathcal{S} .

To give an estimate for the norm of Q , we must use for each triangle T_j its extension \tilde{T}_j .

Theorem 8.4.21 For each $f \in L^p(\Omega)$, $1 \leq p < \infty$, $f \in C(\Omega)$, $p = \infty$, we have the local and global norm estimates

$$\begin{cases} \|Q(f)\|_{p, T_j} \lesssim \|f\|_{p, \tilde{T}_j} \\ \|Q(f)\|_{p, \Omega} \lesssim \|f\|_{p, \Omega} \end{cases}$$

Proof Let $v \in T_j$, we have $Q(f)(v) = \sum_{\nu \in \Lambda_j} \lambda_\nu(f) B_\nu(v)$. Using the fact that $\sum_{\nu \in \Lambda_j} B_\nu \equiv 1$, we get

$$\|Q(f)\|_{p, T_j} \leq \max_{\nu \in \Lambda_j} |\lambda_\nu(f)| \cdot \left\| \sum_{\nu \in \Lambda_j} B_\nu \right\|_{p, T_j} \lesssim \max_{\nu \in \Lambda_j} |T_\nu|^{-p} \|f\|_{p, T_\nu} |T_j|^p \lesssim \|f\|_{p, \tilde{T}_j}$$

For the global norm estimate, we sum over all $\nu \in \Lambda$. □

8.5 Box-splines as finite elements basis

Let $\Omega \subset \mathbb{R}^2$ be a domain for such there is a triangulation Δ such that $\Omega = \cup_{T \in \Delta} hT$, for $h > 0$. In this section, we will take the standard notation for triangulations $\mathcal{T} := \Delta$.

We will denote by

$$\mathcal{S}_h := \left\{ \sum_{\nu \in \mathbb{Z}^2} a_\nu B\left(\frac{\cdot}{h} - \nu | X_n\right); a_\nu \in \mathbb{R}, \nu \in \mathbb{Z}^2 \right\}$$

We introduce the *bent Sobolev spaces* of order $m \in \mathbb{N}$

$$\mathcal{H}^m(\Omega) := \left(\begin{array}{l} f \in L^2(\Omega) \text{ such that} \\ f|_T \in H^m(T) \quad \forall T \in \mathcal{T}_h, \text{ and} \\ D^k f|_{T_1}(v) = D^k f|_{T_2}(v) \quad \forall v \in \partial T_1 \cap \partial T_2, \\ \forall k \in \mathbb{N} \quad 0 \leq k \leq \min\{m_{T_1, T_2}, m-1\}, \\ \forall T_1, T_2 \in \mathcal{T}_h \quad \partial T_1 \cap \partial T_2 \neq \emptyset \end{array} \right)$$

8.5. Box-splines as finite elements basis

where for any $T_1, T_2 \in \mathcal{T}_h$, $\partial T_1 \cap \partial T_2 \neq \emptyset$, we denote by m_{T_1, T_2} the number of continuous derivatives across the edge $\partial T_1 \cap \partial T_2$.

$\mathcal{H}^m(\Omega)$ is a well-defined Hilbert space, endowed with the semi-norms

$$|f|_{l, \Omega}^2 = |f|_{H^l(\Omega)}^2 := \sum_{T \in \mathcal{T}_h} |f|_{H^l(T)}^2 = \sum_{T \in \mathcal{T}_h} |f|_{l, T}^2, \quad \forall 0 \leq l \leq m$$

and norm

$$\|f\|_{m, \Omega}^2 = \|f\|_{H^m(\Omega)}^2 := \sum_{l=0}^m |f|_{l, \Omega}^2$$

8.5.1 Approximation with box-splines

We first, begin by giving an extension of the Bramble-Hilbert lemma over spline spaces.

Lemma 8.5.1 *Let $0 \leq k \leq l \leq p+1$. For each $f \in \mathcal{H}_h^l$, there exists an $s \in \mathcal{S}_h$, such that*

$$|f - s|_{\mathcal{H}_h^k(\tilde{T}_h)} \lesssim h_T^{l-k} |f|_{\mathcal{H}_h^l(\tilde{T}_h)}$$

Proof It is very classic. The proof is inspired from [66] (lemma 3.1) and [59] (theorem 5.5 page 61).

The proof is divided into 3 steps:

- scaling:

Let us consider the function $\hat{f}(\cdot) := f(h_T \cdot)$, we have for every integer l

$$|\hat{f}|_{l, T} = h_T^{-\frac{d}{2}} h_T^l |f|_{l, h_T T}$$

the desired inequality becomes

$$|\hat{f} - \hat{s}|_{\mathcal{H}^k(\tilde{T})} \lesssim |\hat{f}|_{\mathcal{H}^l(\tilde{T})}$$

- Prove that $\|\hat{f} - \hat{s}\|_{p+1} \lesssim |\hat{f}|_{p+1}$.

Let $\mathcal{PP}_{<p+1}$ represent the set of piecewise polynomial functions of degree at most p on \tilde{T} , that is, the set of functions that are polynomials of degree at most p on each element forming \tilde{T} . Notice that $\mathcal{PP}_{<p+1} \cap \mathcal{H}^{p+1}(\tilde{T}) \subset \mathcal{S}_h$.

Let us consider π the $L^2(\tilde{T})$ -projection from $\mathcal{H}^{p+1}(\tilde{T})$ on $\mathcal{PP}_{<p+1} \cap \mathcal{H}^{p+1}(\tilde{T})$. We will prove that the result holds for $\hat{s} := \pi(\hat{f})$.

To this purpose we suppose the contrary: we suppose that we can construct a sequence $\hat{g}_m = \hat{f}_m - \hat{s}_m$ such that $\|\hat{g}_m\|_{p+1} = 1$ and $|\hat{g}_m|_{p+1} \rightarrow 0$.

We then can extract a sequence, also noted \hat{g}_m , and a function g , such that $\|\hat{g}_m - g\|_p \rightarrow 0$ and $\|g\|_p = \lim \|\hat{g}_m\|_{p+1}^2 - |\hat{g}_m|_{p+1}^2 = 1$.

We now prove that $g \in \mathcal{PP}_{<p+1}$:

Let $|\alpha| = p+1$, then

$$\left| \int g \partial^\alpha \varphi \right| = \lim \left| \int \hat{g}_m \partial^\alpha \varphi \right| = \lim \left| \int \partial^\alpha \hat{g}_m \varphi \right| \lesssim \lim |\hat{g}_m|_{p+1}$$

but $\lim |\hat{g}_m|_{p+1} = 0$, then $g \in \mathcal{PP}_{<p+1}$.

We now have to prove that $g = 0$.

We project \hat{f} into $\mathcal{PP}_{<p+1} \cap \mathcal{H}^{p+1}(\tilde{T})$, using an orthonormal basis D_α . We can write :

$$\pi \hat{f}_m = \sum_{|\alpha| \leq p} \langle \hat{f}_m, D_\alpha \rangle D_\alpha$$

8.6. DeRham diagram

and $\hat{g}_m = \hat{f}_m - \hat{s}_m = \hat{f}_m - \pi \hat{f}_m$.

On the other hand: $\langle \hat{f}_m, D_\alpha \rangle_0 = \langle \pi \hat{f}_m, D_\alpha \rangle_0$, $\forall \alpha, |\alpha| \leq p$,

so that, $\forall \alpha, |\alpha| \leq p$: $\langle \hat{g}_m, D_\alpha \rangle_0 = 0$ then $\hat{g}_m = 0$ and by passing to the limit we get $g = 0$.

- The remaining cases $k \leq p$ follow easily.

□

Lemma 8.5.2 *Let $0 \leq k \leq l \leq p + 1$. For each $f \in \mathcal{H}_h^l(\tilde{T}) \cap L^2(\Omega)$ then*

$$|f - Q(f)|_{\mathcal{H}_h^k(T)} \lesssim h_T^{l-k} |f|_{\mathcal{H}_h^l(\tilde{T})}$$

and

$$|Q(f)|_{\mathcal{H}_h^l(T)} \lesssim |f|_{\mathcal{H}_h^l(\tilde{T})}$$

Proof Using the last lemma, we know that for some $s \in \mathcal{S}_h$, we have

$$|f - Q(f)|_{\mathcal{H}_h^k(T)} \leq |f - s|_{\mathcal{H}_h^k(T)} + |Q(s - f)|_{\mathcal{H}_h^k(T)}$$

the first term is bounded using the last lemma. For the second term, we use the inverse inequality for polynomials, which leads to

$$|Q(s - f)|_{\mathcal{H}_h^k(T)} \lesssim h^{-k} \|Q(s - f)\|_{2,T}$$

but, $Q(s - f) = s - f$. We now can use again the last lemma to conclude. □

Lemma 8.5.3 *Let $v_i \in X_n$ be a direction, and $f \in \mathcal{H}_h^l(\tilde{T}) \cap L^2(\Omega)$. We have the following result:*

$$\|\partial_{v_i} Q(f)\|_{p,T_j} \lesssim \|\partial_{v_i} f\|_{p,\tilde{T}_j}$$

Proof First we will need a version of Poincaré's inequality, which is adapted to our case:

$$\forall v_i \in \Omega, \forall \varphi \in \mathcal{C}_0^\infty(\Omega), \quad \|\varphi\|_{0,p,\Omega} \lesssim \|\partial_{v_i} \varphi\|_{0,p,\Omega}.$$

For any polynomial function p_h , we have

$$\|\partial_{v_i} p_h\|_{0,p,T} \leq |p_h|_{1,p,T} \lesssim h_T^{-1} \|p_h\|_{0,p,T}.$$

As $Q(f)$ is a polynomial, then we get by the last Poincaré's inequality

$$\|\partial_{v_i} Q(f)\|_{0,p,T} \lesssim h_T^{-1} \|Q(f)\|_{0,p,T} = h_T^{-1} \|f\|_{0,p,T} \lesssim \|\partial_{v_i} f\|_{0,p,\tilde{T}}.$$

□

8.6 DeRham diagram

In this section, we will consider only periodic boundary condition.

8.6. DeRham diagram

8.6.1 Notations

$X^0 := H^1(\Omega)$, $X^2 := H(\operatorname{div}, \Omega)$, $X^3 := L^2(\Omega)$.

For $h > 0$, let us define

$$\begin{aligned} X_h^0 &:= \operatorname{span} \left\{ B\left(\frac{\cdot}{h} - \nu\right), \quad \nu \in \mathbb{Z}^2 \right\} \\ X_h^2 &:= \operatorname{span} \left\{ \begin{pmatrix} B^y\left(\frac{\cdot}{h} - \nu\right) \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ B^x\left(\frac{\cdot}{h} - \nu\right) \end{pmatrix}, \quad \nu \in \mathbb{Z}^2 \right\} \\ X_h^3 &:= \operatorname{span} \left\{ B^{xy}\left(\frac{\cdot}{h} - \nu\right), \quad \nu \in \mathbb{Z}^2 \right\} \end{aligned}$$

Our goal is to construct the following DeRham diagram

$$\begin{array}{ccccc} & \operatorname{curl} & & \operatorname{div} & \\ X^0 & \longrightarrow & X^2 & \longrightarrow & X^3 \\ \downarrow \pi_h^0 & & \downarrow \pi_h^2 & & \downarrow \pi_h^3 \\ X_h^0 & \longrightarrow & X_h^2 & \longrightarrow & X_h^3 \end{array} \quad (8.6.1)$$

8.6.2 Interpolants and commutativity

Let us define

$$\begin{aligned} \pi_h^0(\varphi) &:= Q_\Lambda^h(\varphi), \quad \varphi \in X^0 \\ \pi_h^2(\mathbf{F})(\mathbf{v}) &:= \begin{pmatrix} \partial_y \pi_h^0 \int_0^{v_y} F_x(v_x, u) du \\ \partial_x \pi_h^0 \int_0^{v_x} F_y(u, v_y) du \end{pmatrix}, \quad \mathbf{F} = (F_x, F_y) \in X^2 \end{aligned}$$

For $\phi \in X^3$, we know that there exists a unique $q \in X^0$, such that $-\Delta q = \phi$, Ω , and $q|_{\partial\Omega} = 0$. Let us take $\mathbf{F} := -\nabla q$, so that we have $\phi = \operatorname{div} \mathbf{F}$. We therefore define π_h^3 as

$$\pi_h^3(\phi) := \pi_h^2(\operatorname{div} \mathbf{F}) = \operatorname{div} \pi_h^2(\mathbf{F})$$

Lemma 8.6.1 (Preserving property) *We have the following properties,*

1. $\pi_h^0(\varphi_h) = \varphi_h$, $\forall \varphi_h \in X_h^0$
2. $\pi_h^2(\mathbf{F}_h) = \mathbf{F}_h$, $\forall \mathbf{F}_h \in X_h^2$
3. $\pi_h^3(\phi_h) = \phi_h$, $\forall \phi_h \in X_h^3$

Proof

1. Let $\varphi_h \in X_h^0$. We can write $\varphi_h = \sum_{\nu \in \mathbb{Z}^2} \phi_h^\nu B\left(\frac{\cdot}{h} - \nu\right)$, so that we have by the preservation property of the quasi-interpolant $\pi_h^0(\varphi_h) = \varphi_h$.

2. Let $\mathbf{F}_h \in X_h^2$, we can write $\mathbf{F}_h = \sum_{\nu \in \mathbb{Z}^2} F_{x,h}^\nu \begin{pmatrix} 0 \\ B^x\left(\frac{\cdot}{h} - \nu\right) \end{pmatrix} + F_{y,h}^\nu \begin{pmatrix} B^y\left(\frac{\cdot}{h} - \nu\right) \\ 0 \end{pmatrix}$.

As π_h^2 is linear, it is sufficient to prove the preserving property for $\begin{pmatrix} 0 \\ B^x\left(\frac{\cdot}{h} - \nu\right) \end{pmatrix}$, $\begin{pmatrix} B^y\left(\frac{\cdot}{h} - \nu\right) \\ 0 \end{pmatrix}$, $\forall \nu \in \mathbb{Z}^2$.

We know that we can write $B^y\left(\frac{\cdot}{h} - \nu\right) = \sum_{\mu \in \Lambda} b^\mu \partial_y B\left(\frac{\cdot}{h} - \mu\right)$, for a unique sequence $\{b^\mu\}_{\mu \in \Lambda}$, which only a finite set is not null. Therefore we have

8.6. DeRham diagram

$$\begin{aligned} \int_0^{v_y} B^y\left(\frac{v_x}{h} - \nu_1, \frac{u}{h} - \nu_2\right) du &= \sum_{\mu \in \Lambda} b^\mu \int_0^{v_y} \partial_y B\left(\frac{v_x}{h} - \mu_1, \frac{u}{h} - \mu_2\right) du \\ &= h \sum_{\mu \in \Lambda} b^\mu \left\{ B\left(\frac{v_x}{h} - \mu_1, \frac{v_y}{h} - \mu_2\right) - B\left(\frac{v_x}{h} - \mu_1, -\mu_2\right) \right\} \end{aligned}$$

By applying π_h^0 , we preserve the last quantity. We now can derive in the \mathbf{e}^2 direction, which leads to

$$\partial_y \pi_h^0 \int_0^{v_y} B^y\left(\frac{v_x}{h} - \nu_1, \frac{u}{h} - \nu_2\right) du = h \sum_{\mu \in \Lambda} b^\mu \left\{ \frac{1}{h} \partial_y B\left(\frac{v_x}{h} - \mu_1, \frac{v_y}{h} - \mu_2\right) - 0 \right\}.$$

which is equal to $B^y\left(\frac{v}{h} - \nu\right)$.

Therefore we have $\pi_h^2 \left(B^y\left(\frac{\dot{\cdot}}{h} - \nu\right) \right) = \left(B^y\left(\frac{\dot{\cdot}}{h} - \nu\right) \right)$. By the same arguments we shall prove $\pi_h^2 \left(B^x\left(\frac{\dot{\cdot}}{h} - \nu\right) \right) = \left(B^x\left(\frac{\dot{\cdot}}{h} - \nu\right) \right)$.

3. Let $\phi_h := \sum_{\nu \in \mathbb{Z}^2} \phi_h^\nu B^{xy}\left(\frac{\dot{\cdot}}{h} - \nu\right) \in X^3$. Let us take $\mathbf{F} := \begin{pmatrix} h \sum_{\nu \in \mathbb{Z}^2} \phi_h^\nu B^y\left(\frac{\dot{\cdot}}{h} - \nu\right) \\ 0 \end{pmatrix}$. It is clear that $\mathbf{F} \in X_h^2$, then $\operatorname{div} \mathbf{F} = \sum_{\nu \in \mathbb{Z}^2} \phi_h^\nu B^{xy}\left(\frac{\dot{\cdot}}{h} - \nu\right) = \phi_h$. Therefore,

$$\pi_h^3(\phi_h) = \operatorname{div}(\pi_h^2 \mathbf{F}) = \operatorname{div}(\mathbf{F}) = \phi_h.$$

□

Lemma 8.6.2 (Commuting property) *We have the following properties,*

1. $\pi_h^2(\operatorname{rot} \varphi) = \operatorname{rot} \pi_h^0(\varphi), \quad \forall \varphi \in X^0$
2. $\pi_h^3(\operatorname{div} \mathbf{F}) = \operatorname{div} \pi_h^2(\mathbf{F}), \quad \forall \mathbf{F} \in X^2$

Proof

1. Let $\varphi \in X^0$. We have

$$\begin{aligned} \pi_h^2(\operatorname{rot} \varphi)(\mathbf{v}) &= \pi_h^2\left(\begin{pmatrix} \partial_y \varphi \\ -\partial_x \varphi \end{pmatrix}\right)(\mathbf{v}) \\ &= \begin{pmatrix} \partial_y \pi_h^0 \int_0^{v_y} \partial_y \varphi(v_x, u) du \\ -\partial_x \pi_h^0 \int_0^{v_x} \partial_x \varphi(u, v_y) du \end{pmatrix} \\ &= \begin{pmatrix} \partial_y \pi_h^0 \varphi \\ -\partial_x \pi_h^0 \varphi \end{pmatrix} \\ &= \operatorname{rot}(\pi_h^0 \varphi) \end{aligned}$$

2. Let $\mathbf{F} \in X^2$, we have by construction $\pi_h^3(\operatorname{div} \mathbf{F}) = \operatorname{div}(\pi_h^2(\mathbf{F}))$.

□

Lemma 8.6.3 (Continuity property) *We have the following properties,*

8.6. DeRham diagram

1. $\|\pi_h^0(\varphi)\|_{L^2(\Omega)} \lesssim \|\varphi\|_{L^2(\Omega)}, \quad \forall \varphi \in X^0$
2. $\|\pi_h^2(\mathbf{F})\|_{(L^2(\Omega))^2} \lesssim \|\mathbf{F}\|_{(L^2(\Omega))^2}, \quad \forall \mathbf{F} \in X^2$
3. $\|\pi_h^3(\phi)\|_{L^2(\Omega)} \lesssim \|\phi\|_{L^2(\Omega)}, \quad \forall \phi \in X^3$

Proof

1. The first assertion is a consequence of theorem (8.4.21).
2. For $\mathbf{F} = \begin{pmatrix} F_x \\ F_y \end{pmatrix} \in X^2$, we have

$$\|\pi_h^2(\mathbf{F})\|_{(L^2(\Omega))^2}^2 = \int_{\Omega} \|\partial_y \pi_h^0 \int_0^{v_y} F_x(v_x, u) du\|_2^2 + \|\partial_x \pi_h^0 \int_0^{v_x} F_y(u, v_y) du\|_2^2$$

let $f_1(v) := \int_0^{v_y} F_x(v_x, u) du$. We know by the first section that

$$\|\partial_y \pi_h^0 f_1\|^2 \leq |\pi_h^0 f_1|_1^2 \lesssim \|\partial_y f_1\|_0^2 = \|F_x\|_0^2$$

we have, if we define $f_2(v) := \int_0^{v_x} F_y(u, v_y) du$:

$$\|\partial_x \pi_h^0 f_2\|^2 \lesssim |\pi_h^0 f_2|_1^2 \lesssim \|\partial_x f_2\|_0^2 = \|F_y\|_0^2$$

then by summing the two inequalities we get

$$\|\pi_h^2(\mathbf{F})\|_{(L^2(\Omega))^2}^2 \lesssim \|\mathbf{F}\|_0^2$$

3. We use the same idea as for the last assertion.

□

8.6.3 Approximation Analysis

Theorem 8.6.4 *We have the following properties,*

1. $|\varphi - \pi_h^0(\varphi)|_{\mathcal{H}^l(T)} \lesssim h_T^{s-l} |\varphi|_{\mathcal{H}^s(\tilde{T})}, \quad \forall \varphi \in X^0 \cap \mathcal{H}^{0,s}(\tilde{T}), \quad 0 \leq l \leq s \leq p+1$
2. $|\mathbf{F} - \pi_h^2(\mathbf{F})|_{\mathcal{H}^l(T)} \lesssim h_T^{s-l} |\mathbf{F}|_{\mathcal{H}^s(\tilde{T})}, \quad \forall \mathbf{F} \in X^2 \cap \mathcal{H}^{2,s}(\tilde{T}), \quad 0 \leq l \leq s \leq p$
3. $|\phi - \pi_h^3(\phi)|_{\mathcal{H}^l(T)} \lesssim h_T^{s-l} |\phi|_{\mathcal{H}^s(\tilde{T})}, \quad \forall \phi \in X^3 \cap \mathcal{H}^{3,s}(\tilde{T}), \quad 0 \leq l \leq s \leq p$

Proof The proof is based on the approximation property (lemma (8.5.1)), and the preserving property of the interpolants:

We know that there exists a spline s such that $|\varphi - s|_{\mathcal{H}_h^k(\tilde{T}_h)} \lesssim h_T^{l-k} |\varphi|_{\mathcal{H}_h^l(\tilde{T}_h)}$, therefore, using the triangle inequality and the preserving property of the interpolant π_h^0 we have

$$|\varphi - \pi_h^0(\varphi)|_{\mathcal{H}^l(T)} \leq |\varphi - s|_{\mathcal{H}^l(T)} + |\pi_h^0(s) - \pi_h^0(\varphi)|_{\mathcal{H}^l(T)} \lesssim h_T^{l-k} |\varphi|_{\mathcal{H}_h^l(\tilde{T}_h)} + |s - \varphi|_{\mathcal{H}^l(T)} \lesssim h_T^{l-k} |\varphi|_{\mathcal{H}_h^l(\tilde{T}_h)}$$

where we used the lemma (8.5.2)

For the other cases, the proof is similar. □

8.7 Boundary Condition using Box-splines

There are several ways to treat the boundary condition. As we notice the definition of Box-splines is similar to the well known uniform B-splines (in $1D$). So how can we force our basis so that it can interpolate (for example) the boundary?

The first solution is inspired by the work done by Hollig ([58, 69]). We can stabilize our basis by introducing an inner and outer Box-splines.

The second solution is to take an adapted box-spline B_{111} for the boundary (figure 8.8) and then define box-spline of high order by the convolution property.

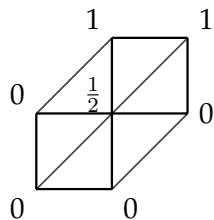


Figure 8.8: Adapted box-spline for the boundary

8.8 Conclusions and Perspectives

In this chapter we gave the premise for the construction of DeRham sequence using a specific splines (Box-splines) defined on triangles. We still have work to do, in order to use interpolating elements for the boundary, to implement the method and verify its numerical accuracy.

Another generalization of this work would be the use of Simplex or Manifold Splines on Delaunay configurations. Such construction was introduced first by Neamtu [87]. More details about the algorithmic aspect can be found in [117]. It will be a real challenge to derive an adequate DeRham sequence for such elements.

Conclusions

In this dissertation, we have tested the IGA approach to solve some problems that arise in Plasma Physics and Electromagnetism. It is of big interest to be able to model and handle these complex geometries that are defined by magnetic field lines (because of the Magnetic Confinement). Most of the work developed during this thesis needs to be improved. Because of time constraint, we have devote much more effort in the development of the library *PyIGA*, which has been written from **scratch** (based only on *pppack*).

Directions for future work

In what follows, suggestions are given for future work and developments on several areas.

Semi-Lagrangian schemes for complex geometries

Handling more complicated mappings than those treated in chapter 6. The implementation of adapted algorithms for the inversion of the grid has to be investigated.

PIC code for complex geometries

In our work, we have rewritten the motions equations on the parametric domain. But this leads to a high computational cost. It will be very important to compare this computational time with the case when particles may live in the physical domain adding inverting the mapping for each particle. It may be also interesting to consider a domain decomposition, using multiple patches, where for a large number of domains we keep the identity mapping, and use *NURBS/B-splines* mapping to locally handle the geometry [38].

IGA in incompressible *MHD*

As seen in the chapter 7, the IGA approach seems very relevant to treat problems that arise in *MHD*. We will surely need to use stabilized elements (*SUPG*, for example). Local refinement, will be of great interest to capture these fine changes of the magnetic structure in magnetic reconnection.

Box-Splines and the DeRham sequence

We would like to finish the study of the DeRham sequence using *Box-Splines*, by taking in account the boundary condition. This will be done by deriving an adequate recurrence formulas for the evaluation of those *Box-splines*.

8.8. Conclusions and Perspectives

As said in the chapter 8, another interesting generalization would be the use of *Simplex-Splines* on Delaunay configurations. We believe that such elements may be interesting for discretizing differential forms.

***GB-Splines* for Maxwell's equations**

In the chapter 4, we have noticed that using *GB-Splines*, we can remove one of the mass matrices involved in the Maxwell's discretized system. We need to derive a DeRham sequence for *GB-Splines*. Following the same idea as for *Box-Splines*, it is not difficult to treat the case of uniform *GB-Splines*, thanks to the **Strang-Fix** theory. A full study must be achieved.

The Fast-IGA solvers

We are investigating new approaches to derive Fast- solvers for IGA. The idea is to use an *unclamped* description of the boundary. We are also developing preconditioners based on the NKP (Near Kronecker Product) problem.

***PyIGA's* perspectives**

The following tasks are under development:

- the use of multi-patches,
- boundary operators,
- the implementation of *GB-Splines*, a fast evaluation algorithm, and appropriate quasi-interpolator,
- setting the boundary condition using the *GUI*,

The following tasks will be done later:

- local refinement using *LR-splines*,
- improving performances: parallelism using either MPI or GPU, and memory use,
- adding routines to treat the free boundary problem, moving boundaries,

Appendix: Decoupling approach for $J(r, \theta)$

The starting equation writes

$$-\nabla_{\perp} \cdot (n_0 \nabla_{\perp} \phi) + [\phi - \langle \phi \rangle] = F, \quad (\text{A.0.1})$$

with $F = (n_i - n_e)/n_0$ and

$$\langle \phi \rangle(r) = \frac{\int \phi(r, \theta, \varphi) J(r, \theta) d\theta d\varphi}{\int J(r, \theta) d\theta d\varphi}.$$

The appendix is devoted to the extension of the decoupling approach to (r, θ) depending jacobians J . As in the case where $J(r, \theta) = r$, we integrate (A.0.1) with respect to θ, φ to get a one-dimensional equation on $\bar{\phi}(r) = 1/(4\pi^2) \int \phi d\theta d\varphi$

$$-\nabla_{\perp} \cdot (n_0 \nabla_{\perp} \bar{\phi}) + \bar{\phi} - \langle \phi \rangle = \bar{F}. \quad (\text{A.0.2})$$

Let us remark that this operation is transparent for the operator $\langle \cdot \rangle$. Moreover, we are looking for an equation satisfied by $\Phi = \phi - \bar{\phi}$ from (A.0.1):

$$\begin{aligned} -\nabla_{\perp} \cdot (n_0 \nabla_{\perp} \Phi) - \nabla_{\perp} \cdot (n_0 \nabla_{\perp} \bar{\phi}) + \phi - \bar{\phi} + \bar{\phi} - \langle \phi \rangle &= F \\ -\nabla_{\perp} \cdot (n_0 \nabla_{\perp} \Phi) - \nabla_{\perp} \cdot (n_0 \nabla_{\perp} \bar{\phi}) + \Phi + \bar{\phi} - \langle \phi \rangle &= F \\ -\nabla_{\perp} \cdot (n_0 \nabla_{\perp} \Phi) + \bar{F} - \bar{\phi} + \langle \phi \rangle + \Phi + \bar{\phi} - \langle \phi \rangle &= F \text{ thanks to (A.0.2)} \\ -\nabla_{\perp} \cdot (n_0 \nabla_{\perp} \Phi) + \bar{F} - \bar{\phi} + \langle \phi \rangle + \Phi + \bar{\phi} - \langle \phi \rangle &= F \\ -\nabla_{\perp} \cdot (n_0 \nabla_{\perp} \Phi) + \Phi &= F - \bar{F}. \end{aligned} \quad (\text{A.0.3})$$

Let us come back to (A.0.2) in order to derive an equation satisfied by $\langle \phi \rangle$. From (A.0.2), we have

$$-\nabla_{\perp} \cdot (n_0 \nabla_{\perp} (\bar{\phi} - \langle \phi \rangle)) - \nabla_{\perp} \cdot (n_0 \nabla_{\perp} \langle \phi \rangle) + \bar{\phi} - \langle \phi \rangle = \bar{F}.$$

By introducing the notation $h(r) = \bar{\phi} - \langle \phi \rangle$, we can derive an equation for $\langle \phi \rangle(r)$

$$-\nabla_{\perp} \cdot (n_0 \nabla_{\perp} \langle \phi \rangle) = \bar{F} + \nabla_{\perp} \cdot (n_0 \nabla_{\perp} h) - h, \quad h(r) = \bar{\phi} - \langle \phi \rangle. \quad (\text{A.0.4})$$

To conclude, we present the decoupling algorithm which enables to solve (A.0.1)

- (a) solve (A.0.3) $\rightarrow \Phi = \phi - \bar{\phi}$
- (b) compute $\langle \Phi \rangle$

-
- (c) compute (and store) $\text{tmp} = (\Phi - \langle \Phi \rangle)$
 - (d) compute $h(r) = 1/(4\pi^2) \int (\Phi - \langle \Phi \rangle) d\theta d\varphi = \bar{\phi} - \langle \phi \rangle$
 - (e) solve (A.0.4) $\rightarrow \langle \phi \rangle$
 - (f) $\phi = \text{tmp} + \langle \phi \rangle = (\Phi - \langle \Phi \rangle) + \langle \phi \rangle = \phi - \bar{\phi} - \langle \phi \rangle + \bar{\phi} + \langle \phi \rangle$ using (c) and (e).

Remark A.0.1 One of the key argument comes from the fact that operators $\langle \cdot \rangle$ and $\bar{\cdot}$ are transparent from each other

$$\langle \bar{\phi} \rangle = \int \bar{\phi} J d\theta d\varphi / \left(\int J d\theta d\varphi \right) = \bar{\phi} \int J d\theta d\varphi / \left(\int J d\theta d\varphi \right) = \bar{\phi},$$

and

$$\langle \bar{\phi} \rangle = 1/(4\pi^2) \int \langle \phi \rangle d\theta d\varphi = \langle \phi \rangle / (4\pi^2) \int d\theta d\varphi = \langle \phi \rangle.$$

Transformation compatible with grad, div and curl operators

In order to define our basis functions on the parametric domain, which is a rectangular domain of \mathbb{R}^2 with cartesian coordinates and then to map them onto a patch of the physical domain, we need to define a transformation of scalar and vector fields which is compatible with our differential operators (grad, div and curl). This is provided to us by the pullback operator for differential forms which is designed to commute with the exterior derivative. Hence compatible transformations will be provided to us by associating our scalar or vector fields to a well chosen differential form and using the pullback.

In our case, we have differential forms defined on Q and need to construct the associated differential forms on K . For this we need a C^1 diffeomorphism $G : K \rightarrow Q$. Let us recall the pullback formula for 0, 1 and 2-forms. A 0-form on Q is a function (or a scalar field) $\varphi(\xi, \eta)$. The pullback of φ on K is in this case simply $\varphi = \varphi \circ G$.

A 1-form on the parametric space can be written $\underline{\omega} = \omega_1(\xi, \eta)d\xi + \omega_2(\xi, \eta)d\eta$. Denoting by G_1 and G_2 the components of the diffeomorphism G , the pullback of $\underline{\omega}$ on K is then defined by

$$\begin{aligned} \omega &= G_*\underline{\omega} = \omega_1 \circ G dG_1 + \omega_2 \circ G dG_2 \\ &= \left(\omega_1 \frac{\partial G_1}{\partial x} + \omega_2 \frac{\partial G_2}{\partial x} \right) dx + \left(\omega_1 \frac{\partial G_1}{\partial y} + \omega_2 \frac{\partial G_2}{\partial y} \right) dy, \end{aligned} \quad (\text{B.0.1})$$

where we denote by $\omega_1(x, y) = \omega_1 \circ G(x, y)$ and $\omega_2(x, y) = \omega_2 \circ G(x, y)$.

A 2-form on the parametric space can be written $\underline{\sigma}(\xi, \eta) d\xi \wedge d\eta$ and its pullback on K by the diffeomorphism G is defined by

$$\begin{aligned} \sigma(x, y) dx \wedge dy &= G_*(\underline{\sigma} d\xi \wedge d\eta) = \underline{\sigma} \circ G dG_1 \wedge dG_2 \\ &= \sigma(x, y) \left(\frac{\partial G_1}{\partial x} \frac{\partial G_2}{\partial y} - \frac{\partial G_2}{\partial x} \frac{\partial G_1}{\partial y} \right) dx \wedge dy, \end{aligned} \quad (\text{B.0.2})$$

where we denote by $\sigma(x, y) = \underline{\sigma} \circ G(x, y)$.

Now a vector field in a 2D space is associated to a differential 1-form. This 1-form depends on the sequence of spaces we are working on and is chosen such that its exterior derivative corresponds either to the curl or the divergence of the vector field. Note that in both cases a function φ (or scalar field) can be associated to either a 0-form which is the function itself or the two form $\varphi d\xi \wedge d\eta$.

Let us start with the case of (4.3.9). In this case the exterior derivative of a 0-form should be associated to the grad operator and the exterior derivative of a 1-form should

be associated to the curl operator. This is the case if we associate a generic vector field $\underline{\Psi}(\xi, \eta) = (\underline{\Psi}^{(1)}(\xi, \eta), \underline{\Psi}^{(2)}(\xi, \eta))^T$ to the differential form

$$\underline{\omega}_c = \underline{\Psi}^{(1)}(\xi, \eta) d\xi + \underline{\Psi}^{(2)}(\xi, \eta) d\eta.$$

Indeed, take a function (or 0-form) $\underline{\varphi}$. Then, on the one hand

$$\mathbf{grad} \underline{\varphi} = (\partial_\xi \underline{\varphi}, \partial_\eta \underline{\varphi})^T$$

which is associated to the one form

$$\partial_\xi \underline{\varphi} d\xi + \partial_\eta \underline{\varphi} d\eta = d\underline{\varphi},$$

an on the other hand

$$\mathbf{rot} \underline{\Psi}(\xi, \eta) d\xi \wedge d\eta = (\partial_\xi \underline{\Psi}^{(2)} - \partial_\eta \underline{\Psi}^{(1)}) d\xi \wedge d\eta = d\underline{\omega}_c.$$

Let us now consider the case of (4.3.10). Then the exterior derivative of a 0-form should be associated to the **rot** operator and the exterior derivative of a 1-form should be associated to the **div** operator. This is the case if we associate a generic vector field $\underline{\Psi}(\xi, \eta) = (\underline{\Psi}^{(1)}(\xi, \eta), \underline{\Psi}^{(2)}(\xi, \eta))^T$ to the differential form

$$\underline{\omega}_d = \underline{\Psi}^{(1)}(\xi, \eta) d\eta - \underline{\Psi}^{(2)}(\xi, \eta) d\xi.$$

Indeed, take a function (or 0-form) $\underline{\varphi}$. Then, on the one hand

$$\mathbf{rot} \underline{\varphi} = (\partial_\eta \underline{\varphi}, -\partial_\xi \underline{\varphi})^T$$

which is associated to the one form

$$\partial_\eta \underline{\varphi} d\eta - (-\partial_\xi \underline{\varphi} d\xi) = d\underline{\varphi},$$

an on the other hand

$$\mathbf{div} \underline{\Psi}(\xi, \eta) d\xi \wedge d\eta = (\partial_\xi \underline{\Psi}^{(1)} + \partial_\eta \underline{\Psi}^{(2)}) d\xi \wedge d\eta = d\underline{\omega}_d.$$

Having now associated our functions and vector fields to differential forms, we can use the expression of the pullbacks to define the adequate scalar and vector field transformations.

In particular, when using the spaces associated to (4.3.10). We need to transform the basis functions associated to V , this is straightforward as they are functions associated to 0-forms, and to W_{div} . To defined the transformation of a vector field $\underline{\Psi}(\xi, \eta) = (\underline{\Psi}^{(1)}(\xi, \eta), \underline{\Psi}^{(2)}(\xi, \eta))^T \in W_{div}$ we use the pullback formula (B.0.1) for the 1-form $\underline{\omega}_d = \underline{\Psi}^{(1)}(\xi, \eta) d\eta - \underline{\Psi}^{(2)}(\xi, \eta) d\xi$. This yields

$$\omega_d = \left(-\underline{\Psi}^{(2)} \circ G \frac{\partial G_1}{\partial x} + \underline{\Psi}^{(1)} \circ G \frac{\partial G_2}{\partial x} \right) dx + \left(-\underline{\Psi}^{(2)} \circ G \frac{\partial G_1}{\partial y} + \underline{\Psi}^{(1)} \circ G \frac{\partial G_2}{\partial y} \right) dy,$$

which is associated to the vector field

$$\Psi = \left(\left(-\underline{\Psi}^{(2)} \circ G \frac{\partial G_1}{\partial y} + \underline{\Psi}^{(1)} \circ G \frac{\partial G_2}{\partial y} \right), \left(\underline{\Psi}^{(2)} \circ G \frac{\partial G_1}{\partial x} - \underline{\Psi}^{(1)} \circ G \frac{\partial G_2}{\partial x} \right) \right)^T. \quad (\text{B.0.3})$$

Equations of motion

C.1 Cylindrical coordinates

Change of coordinates

Let

$$\mathbf{X} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

be the cartesian coordinates and

$$\mathbf{q} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} r \\ \theta \\ z \end{pmatrix}$$

be the cylindrical ones, with

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} r \cos \theta \\ r \sin \theta \\ z \end{pmatrix} \Leftrightarrow \begin{pmatrix} r \\ \theta \\ z \end{pmatrix} = \begin{pmatrix} \sqrt{x^2 + y^2} \\ \arctan(\frac{y}{x}) \\ z \end{pmatrix}.$$

The jacobian of this transformation writes:

$$J = \det \begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \theta} & \frac{\partial x}{\partial z} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \theta} & \frac{\partial y}{\partial z} \\ \frac{\partial z}{\partial r} & \frac{\partial z}{\partial \theta} & \frac{\partial z}{\partial z} \end{pmatrix} = \det \begin{pmatrix} \cos \theta & -r \sin \theta & 0 \\ \sin \theta & r \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} = r.$$

The Lagrangian

In cartesian coordinates, the Lagrangian writes,

$$L(\mathbf{X}, \dot{\mathbf{X}}, t) = \frac{1}{2} m \dot{\mathbf{X}}^2 - e(\mathbf{A} \cdot \dot{\mathbf{X}} - \phi) = \frac{1}{2} m(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) - e(A_x \dot{x} + A_y \dot{y} + A_z \dot{z} - \phi),$$

where,

$$\dot{u} = \frac{du}{dt},$$

C.1. Cylindrical coordinates

with \mathbf{A} the potential vector, and ϕ scalar potential.

Let us search for its expression in cylindrical coordinates,

$$\tilde{L}(\mathbf{q}, \dot{\mathbf{q}}, t) = L(\mathbf{X}(\mathbf{q}), \dot{\mathbf{X}}(\mathbf{q}), t).$$

We have :

$$\begin{aligned} \dot{x} &= \frac{dx}{dt} = \frac{\partial x}{\partial r} \frac{\partial r}{\partial t} + \frac{\partial x}{\partial \theta} \frac{\partial \theta}{\partial t} + \frac{\partial x}{\partial z} \frac{\partial z}{\partial t} \\ \dot{x} &= \dot{r} \cos \theta - r \dot{\theta} \sin \theta & \Rightarrow \dot{x}^2 &= \dot{r}^2 \cos^2 \theta + r^2 \dot{\theta}^2 \sin^2 \theta - 2r \dot{r} \dot{\theta} \cos \theta \sin \theta \end{aligned}$$

and

$$\dot{y} = \dot{r} \sin \theta + r \dot{\theta} \cos \theta \quad \Rightarrow \dot{y}^2 = \dot{r}^2 \sin^2 \theta + r^2 \dot{\theta}^2 \cos^2 \theta + 2r \dot{r} \dot{\theta} \cos \theta \sin \theta$$

and

$$\dot{z} = \dot{z} \quad \Rightarrow \dot{z}^2 = \dot{z}^2.$$

Therefore,

$$\dot{x}^2 + \dot{y}^2 + \dot{z}^2 = \dot{r}^2 + r^2 \dot{\theta}^2 + \dot{z}^2.$$

Covariant transformation of the potential vector \mathbf{A}

We have,

$$\begin{aligned} A_x &= \frac{\partial r}{\partial x} A_r + \frac{\partial \theta}{\partial x} A_\theta + \frac{\partial z}{\partial x} A_z \\ &= \frac{2x}{2\sqrt{x^2 + y^2}} A_r + \frac{-y}{x^2 + y^2} A_\theta \\ &= \frac{r \cos \theta}{r} A_r - \frac{r \sin \theta}{r^2} A_\theta \\ &= \cos \theta A_r - \frac{\sin \theta}{r} A_\theta, \end{aligned}$$

and,

$$A_y = \sin \theta A_r + \frac{\cos \theta}{r} A_\theta$$

and,

$$A_z = \frac{\partial r}{\partial z} A_r + \frac{\partial \theta}{\partial z} A_\theta + \frac{\partial z}{\partial z} A_z = A_z.$$

C.1. Cylindrical coordinates

Hence, we get,

$$\begin{aligned} A_x \dot{x} + A_y \dot{y} + A_z \dot{z} &= (\cos \theta A_r - \frac{\sin \theta}{r} A_\theta)(\dot{r} \cos \theta - r \dot{\theta} \sin \theta) \\ &\quad + (\sin \theta A_r + \frac{\cos \theta}{r} A_\theta)(\dot{r} \sin \theta + r \dot{\theta} \cos \theta) + A_z \dot{z} \\ &= A_r \dot{r} + A_\theta \dot{\theta} + A_z \dot{z} \end{aligned}$$

and,

$$\tilde{L} = \frac{1}{2} m (\dot{r}^2 + r^2 \dot{\theta}^2 + \dot{z}^2) - e(A_r \dot{r} + A_\theta \dot{\theta} + A_z \dot{z} - \phi).$$

Euler-Lagrange equations

Let us consider the Euler-Lagrange equations:

$$\frac{d}{dt} \frac{\partial \tilde{L}}{\partial \dot{\mathbf{q}}} = \frac{\partial \tilde{L}}{\partial \mathbf{q}}.$$

as,

$$\begin{aligned} \frac{\partial \tilde{L}}{\partial r} &= m r \dot{\theta}^2 - e \left(\dot{r} \frac{\partial A_r}{\partial r} + \dot{\theta} \frac{\partial A_\theta}{\partial r} + \dot{z} \frac{\partial A_z}{\partial r} - \frac{\partial \phi}{\partial r} \right), \\ \frac{\partial \tilde{L}}{\partial \dot{\theta}} &= -e \left(\dot{r} \frac{\partial A_r}{\partial \theta} + \dot{\theta} \frac{\partial A_\theta}{\partial \theta} + \dot{z} \frac{\partial A_z}{\partial \theta} - \frac{\partial \phi}{\partial \theta} \right), \\ \frac{\partial \tilde{L}}{\partial z} &= -e \left(\dot{r} \frac{\partial A_r}{\partial z} + \dot{\theta} \frac{\partial A_\theta}{\partial z} + \dot{z} \frac{\partial A_z}{\partial z} - \frac{\partial \phi}{\partial z} \right), \end{aligned}$$

and,

$$\begin{aligned} \frac{\partial \tilde{L}}{\partial \dot{r}} &= m \dot{r} - e A_r, \\ \frac{\partial \tilde{L}}{\partial \dot{\theta}} &= m r^2 \dot{\theta} - e A_\theta, \\ \frac{\partial \tilde{L}}{\partial \dot{z}} &= m \dot{z} - e A_z, \end{aligned}$$

we get,

$$\begin{aligned} \frac{d}{dt} \frac{\partial \tilde{L}}{\partial \dot{r}} &= m \ddot{r} - e \left(\dot{r} \frac{\partial A_r}{\partial r} + \dot{\theta} \frac{\partial A_r}{\partial \theta} + \dot{z} \frac{\partial A_r}{\partial z} + \frac{\partial A_r}{\partial t} \right), \\ \frac{d}{dt} \frac{\partial \tilde{L}}{\partial \dot{\theta}} &= 2 m r \dot{r} \dot{\theta} + m r^2 \ddot{\theta} - e \left(\dot{r} \frac{\partial A_\theta}{\partial r} + \dot{\theta} \frac{\partial A_\theta}{\partial \theta} + \dot{z} \frac{\partial A_\theta}{\partial z} + \frac{\partial A_\theta}{\partial t} \right), \\ \frac{d}{dt} \frac{\partial \tilde{L}}{\partial \dot{z}} &= m \ddot{z} - e \left(\dot{r} \frac{\partial A_z}{\partial r} + \dot{\theta} \frac{\partial A_z}{\partial \theta} + \dot{z} \frac{\partial A_z}{\partial z} + \frac{\partial A_z}{\partial t} \right). \end{aligned}$$

Therefore, we get the motion equations:

C.1. Cylindrical coordinates

$$\left\{ \begin{array}{l} m\ddot{r} - e\left(\dot{r}\frac{\partial A_r}{\partial r} + \dot{\theta}\frac{\partial A_r}{\partial \theta} + \dot{z}\frac{\partial A_r}{\partial z} + \frac{\partial A_r}{\partial t}\right) = mr\dot{\theta}^2 - e\left(\dot{r}\frac{\partial A_r}{\partial r} + \dot{\theta}\frac{\partial A_\theta}{\partial r} + \dot{z}\frac{\partial A_z}{\partial r} - \frac{\partial \phi}{\partial r}\right) \\ 2mr\dot{r}\dot{\theta} + mr^2\ddot{\theta} - e\left(\dot{r}\frac{\partial A_\theta}{\partial r} + \dot{\theta}\frac{\partial A_\theta}{\partial \theta} + \dot{z}\frac{\partial A_\theta}{\partial z} + \frac{\partial A_\theta}{\partial t}\right) = -e\left(\dot{r}\frac{\partial A_r}{\partial \theta} + \dot{\theta}\frac{\partial A_\theta}{\partial \theta} + \dot{z}\frac{\partial A_z}{\partial \theta} - \frac{\partial \phi}{\partial \theta}\right) \\ m\ddot{z} - e\left(\dot{r}\frac{\partial A_z}{\partial r} + \dot{\theta}\frac{\partial A_z}{\partial \theta} + \dot{z}\frac{\partial A_z}{\partial z} + \frac{\partial A_z}{\partial t}\right) = -e\left(\dot{r}\frac{\partial A_r}{\partial z} + \dot{\theta}\frac{\partial A_\theta}{\partial z} + \dot{z}\frac{\partial A_z}{\partial z} - \frac{\partial \phi}{\partial z}\right) \end{array} \right.$$

\Leftrightarrow

$$\left\{ \begin{array}{l} m\ddot{r} - e\left(\dot{\theta}\frac{\partial A_r}{\partial \theta} + \dot{z}\frac{\partial A_r}{\partial z} + \frac{\partial A_r}{\partial t}\right) = mr\dot{\theta}^2 - e\left(\dot{\theta}\frac{\partial A_\theta}{\partial r} + \dot{z}\frac{\partial A_z}{\partial r} - \frac{\partial \phi}{\partial r}\right) \\ 2mr\dot{r}\dot{\theta} + mr^2\ddot{\theta} - e\left(\dot{r}\frac{\partial A_\theta}{\partial r} + \dot{z}\frac{\partial A_\theta}{\partial z} + \frac{\partial A_\theta}{\partial t}\right) = -e\left(\dot{r}\frac{\partial A_r}{\partial \theta} + \dot{z}\frac{\partial A_z}{\partial \theta} - \frac{\partial \phi}{\partial \theta}\right) \\ m\ddot{z} - e\left(\dot{r}\frac{\partial A_z}{\partial r} + \dot{\theta}\frac{\partial A_z}{\partial \theta} + \frac{\partial A_z}{\partial t}\right) = -e\left(\dot{r}\frac{\partial A_r}{\partial z} + \dot{\theta}\frac{\partial A_\theta}{\partial z} - \frac{\partial \phi}{\partial z}\right) \end{array} \right.$$

\Leftrightarrow

$$\left\{ \begin{array}{l} m(\ddot{r} - r\dot{\theta}^2) = -e\left(\dot{\theta}\left(\frac{\partial A_\theta}{\partial r} - \frac{\partial A_r}{\partial \theta}\right) + \dot{z}\left(\frac{\partial A_z}{\partial r} - \frac{\partial A_r}{\partial z}\right) - \frac{\partial A_r}{\partial t} - \frac{\partial \phi}{\partial r}\right) \\ 2mr\dot{r}\dot{\theta} + mr^2\ddot{\theta} = -e\left(\dot{r}\left(\frac{\partial A_r}{\partial \theta} - \frac{\partial A_\theta}{\partial r}\right) + \dot{z}\left(\frac{\partial A_z}{\partial \theta} - \frac{\partial A_\theta}{\partial z}\right) - \frac{\partial A_\theta}{\partial t} - \frac{\partial \phi}{\partial \theta}\right) \\ m\ddot{z} = -e\left(\dot{r}\left(\frac{\partial A_r}{\partial z} - \frac{\partial A_z}{\partial r}\right) + \dot{\theta}\left(\frac{\partial A_\theta}{\partial z} - \frac{\partial A_z}{\partial \theta}\right) - \frac{\partial A_z}{\partial t} - \frac{\partial \phi}{\partial z}\right) \end{array} \right.$$

\Leftrightarrow

$$\left(\begin{array}{c} m(\ddot{r} - r\dot{\theta}^2) \\ 2mr\dot{r}\dot{\theta} + mr^2\ddot{\theta} \\ m\ddot{z} \end{array} \right) = -e(\dot{\mathbf{q}} \wedge (\nabla \wedge \mathbf{A}) - \frac{\partial \mathbf{A}}{\partial t} - \nabla \phi)$$

\Leftrightarrow

$$\left(\begin{array}{c} \ddot{z} \\ \ddot{r} - r\dot{\theta}^2 \\ 2r\dot{r}\dot{\theta} + r^2\ddot{\theta} \end{array} \right) = -\frac{e}{m} \left(\left(\begin{array}{c} \dot{z} \\ \dot{r} \\ \dot{\theta} \end{array} \right) \wedge \left(\begin{array}{c} r B_z \\ r B_r \\ \frac{B_\theta}{r} \end{array} \right) + \left(\begin{array}{c} E_z \\ E_r \\ E_\theta \end{array} \right) \right)$$

\Leftrightarrow

C.2. General coordinates

$$\begin{pmatrix} r \ddot{z} \\ r \ddot{r} - (r\dot{\theta})^2 \\ 2\dot{r}\dot{\theta} + r\ddot{\theta} \end{pmatrix} = -\frac{e}{m} \left(\begin{pmatrix} \dot{z} \\ \dot{r} \\ \dot{\theta}r^2 \end{pmatrix} \wedge \begin{pmatrix} B_z \\ B_r \\ B_\theta \end{pmatrix} + \begin{pmatrix} r E_z \\ r E_r \\ \frac{E_\theta}{r} \end{pmatrix} \right).$$

C.2 General coordinates

We would like now to go from any general coordinates system (ξ, η, θ) , to cylindrical coordinates. Let \mathbf{F} be a mapping such that :

$$\mathbf{F}(\xi, \eta, \theta) = (z, r, \theta) \quad (\text{C.2.1})$$

and we assume a $2D$ axisymmetric geometry.

Under the assumption $\dot{\theta} = 0$, we have :

$$\mathbf{F}(\xi, \eta) = (z, r).$$

We recall the expression of the Lagrangian in cylindrical coordinates,

$$\tilde{L} = \frac{1}{2}m(\dot{z}^2 + \dot{r}^2) - e(A_z \dot{z} + A_r \dot{r} - \phi).$$

We have,

$$\dot{r} = \frac{dr}{dt} = \frac{\partial r}{\partial \xi} \frac{\partial \xi}{\partial t} + \frac{\partial r}{\partial \eta} \frac{\partial \eta}{\partial t} = \frac{\partial r}{\partial \xi} \dot{\xi} + \frac{\partial r}{\partial \eta} \dot{\eta},$$

and,

$$\dot{z} = \frac{\partial z}{\partial \xi} \dot{\xi} + \frac{\partial z}{\partial \eta} \dot{\eta}.$$

therefor,

$$\dot{r}^2 + \dot{z}^2 = \left(\left(\frac{\partial r}{\partial \xi} \right)^2 + \left(\frac{\partial z}{\partial \xi} \right)^2 \right) \dot{\xi}^2 + \left(\left(\frac{\partial r}{\partial \eta} \right)^2 + \left(\frac{\partial z}{\partial \eta} \right)^2 \right) \dot{\eta}^2 + 2 \left(\frac{\partial r}{\partial \xi} \frac{\partial r}{\partial \eta} + \frac{\partial z}{\partial \xi} \frac{\partial z}{\partial \eta} \right) \dot{\xi} \dot{\eta}.$$

Covariant transformation of the potential vector \mathbf{A}

We have,

$$A_\xi = \frac{\partial r}{\partial \xi} A_r + \frac{\partial z}{\partial \xi} A_z,$$

and,

$$A_\eta = \frac{\partial r}{\partial \eta} A_r + \frac{\partial z}{\partial \eta} A_z.$$

C.2. General coordinates

hence, in the new coordinates, the potential vector writes,

$$\begin{aligned} A_r \dot{r} + A_z \dot{z} &= A_r \left(\frac{\partial r}{\partial \xi} \dot{\xi} + \frac{\partial r}{\partial \eta} \dot{\eta} \right) + A_z \left(\frac{\partial z}{\partial \xi} \dot{\xi} + \frac{\partial z}{\partial \eta} \dot{\eta} \right) \\ &= \left(A_r \frac{\partial r}{\partial \xi} + A_z \frac{\partial z}{\partial \xi} \right) \dot{\xi} + \left(A_r \frac{\partial r}{\partial \eta} + A_z \frac{\partial z}{\partial \eta} \right) \dot{\eta} \\ &= A_\xi \dot{\xi} + A_\eta \dot{\eta}, \end{aligned}$$

so that,

$$\begin{aligned} L &= \frac{1}{2} m \left(\left(\frac{\partial r}{\partial \xi} \right)^2 + \left(\frac{\partial z}{\partial \xi} \right)^2 \right) \dot{\xi}^2 + \left(\left(\frac{\partial r}{\partial \eta} \right)^2 + \left(\frac{\partial z}{\partial \eta} \right)^2 \right) \dot{\eta}^2 + 2 \left(\frac{\partial r}{\partial \xi} \frac{\partial r}{\partial \eta} + \frac{\partial z}{\partial \xi} \frac{\partial z}{\partial \eta} \right) \dot{\xi} \dot{\eta} \\ &\quad - e(A_\xi \dot{\xi} + A_\eta \dot{\eta} - \phi). \end{aligned}$$

Let us denote,

$$M_\xi = \left(\frac{\partial r}{\partial \xi} \right)^2 + \left(\frac{\partial z}{\partial \xi} \right)^2,$$

$$M_\eta = \left(\frac{\partial r}{\partial \eta} \right)^2 + \left(\frac{\partial z}{\partial \eta} \right)^2,$$

and,

$$M_{\xi\eta} = \frac{\partial r}{\partial \xi} \frac{\partial r}{\partial \eta} + \frac{\partial z}{\partial \xi} \frac{\partial z}{\partial \eta},$$

we can now write,

$$L = \frac{1}{2} m (M_\xi \dot{\xi}^2 + M_\eta \dot{\eta}^2 + 2M_{\xi\eta} \dot{\xi} \dot{\eta}) - e(A_\xi \dot{\xi} + A_\eta \dot{\eta} - \phi).$$

Euler-Lagrange equations

We recall that,

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{\mathbf{q}}} = \frac{\partial L}{\partial \mathbf{q}},$$

where,

$$\mathbf{q} = \begin{pmatrix} \xi \\ \eta \end{pmatrix}.$$

Let us compute the following partial derivatives:

C.2. General coordinates

$$\frac{\partial M_\xi}{\partial \xi} = 2 \frac{\partial^2 r}{\partial \xi^2} \frac{\partial r}{\partial \xi} + 2 \frac{\partial^2 z}{\partial \xi^2} \frac{\partial z}{\partial \xi},$$

$$\frac{\partial M_\xi}{\partial \eta} = 2 \frac{\partial^2 r}{\partial \eta \partial \xi} \frac{\partial r}{\partial \xi} + 2 \frac{\partial^2 z}{\partial \eta \partial \xi} \frac{\partial z}{\partial \xi},$$

$$\frac{\partial M_\eta}{\partial \eta} = 2 \frac{\partial^2 r}{\partial \eta^2} \frac{\partial r}{\partial \eta} + 2 \frac{\partial^2 z}{\partial \eta^2} \frac{\partial z}{\partial \eta},$$

$$\frac{\partial M_\eta}{\partial \xi} = 2 \frac{\partial^2 r}{\partial \eta \partial \xi} \frac{\partial r}{\partial \eta} + 2 \frac{\partial^2 z}{\partial \eta \partial \xi} \frac{\partial z}{\partial \eta},$$

$$\frac{\partial M_{\xi\eta}}{\partial \xi} = \frac{\partial^2 r}{\partial \xi^2} \frac{\partial r}{\partial \eta} + \frac{\partial^2 r}{\partial \eta \partial \xi} \frac{\partial r}{\partial \xi} + \frac{\partial^2 z}{\partial \xi^2} \frac{\partial z}{\partial \eta} + \frac{\partial^2 z}{\partial \eta \partial \xi} \frac{\partial z}{\partial \xi},$$

and,

$$\frac{\partial M_{\xi\eta}}{\partial \eta} = \frac{\partial^2 r}{\partial \eta \partial \xi} \frac{\partial r}{\partial \eta} + \frac{\partial^2 r}{\partial \eta^2} \frac{\partial r}{\partial \xi} + \frac{\partial^2 z}{\partial \eta \partial \xi} \frac{\partial z}{\partial \eta} + \frac{\partial^2 z}{\partial \eta^2} \frac{\partial z}{\partial \xi}.$$

hence, we get,

$$\frac{\partial L}{\partial \xi} = \frac{1}{2} m \left(\frac{\partial M_\xi}{\partial \xi} \dot{\xi}^2 + \frac{\partial M_\eta}{\partial \xi} \dot{\eta}^2 + 2 \frac{\partial M_{\xi\eta}}{\partial \xi} \dot{\xi} \dot{\eta} \right) - e \left(\frac{\partial A_\xi}{\partial \xi} \dot{\xi} + \frac{\partial A_\eta}{\partial \xi} \dot{\eta} - \frac{\partial \phi}{\partial \xi} \right),$$

$$\frac{\partial L}{\partial \eta} = \frac{1}{2} m \left(\frac{\partial M_\xi}{\partial \eta} \dot{\xi}^2 + \frac{\partial M_\eta}{\partial \eta} \dot{\eta}^2 + 2 \frac{\partial M_{\xi\eta}}{\partial \eta} \dot{\xi} \dot{\eta} \right) - e \left(\frac{\partial A_\xi}{\partial \eta} \dot{\xi} + \frac{\partial A_\eta}{\partial \eta} \dot{\eta} - \frac{\partial \phi}{\partial \eta} \right).$$

But, as we have,

$$\frac{\partial L}{\partial \dot{\xi}} = \frac{1}{2} m (2M_\xi \dot{\xi} + 2M_{\xi\eta} \dot{\eta}) - eA_\xi = m(M_\xi \dot{\xi} + M_{\xi\eta} \dot{\eta}) - eA_\xi,$$

$$\frac{\partial L}{\partial \dot{\eta}} = \frac{1}{2} m (2M_\eta \dot{\eta} + 2M_{\xi\eta} \dot{\xi}) - eA_\eta = m(M_\eta \dot{\eta} + M_{\xi\eta} \dot{\xi}) - eA_\eta.$$

The time derivatives writes,

C.2. General coordinates

$$\begin{aligned} \frac{d}{dt} \frac{\partial L}{\partial \dot{\xi}} &= m(M_\xi \ddot{\xi} + M_{\xi\eta} \ddot{\eta} + \frac{\partial M_\xi}{\partial \xi} (\dot{\xi})^2 + \frac{\partial M_\xi}{\partial \eta} \dot{\eta} \dot{\xi} + \frac{\partial M_{\xi\eta}}{\partial \eta} (\dot{\eta})^2 + \frac{\partial M_{\xi\eta}}{\partial \xi} \dot{\xi} \dot{\eta}) \\ &\quad - e \left(\frac{\partial A_\xi}{\partial \xi} \dot{\xi} + \frac{\partial A_\xi}{\partial \eta} \dot{\eta} + \frac{\partial A_\xi}{\partial t} \right), \end{aligned}$$

$$\begin{aligned} \frac{d}{dt} \frac{\partial L}{\partial \dot{\eta}} &= m(M_\eta \ddot{\eta} + M_{\xi\eta} \ddot{\xi} + \frac{\partial M_{\xi\eta}}{\partial \xi} (\dot{\xi})^2 + \frac{\partial M_{\xi\eta}}{\partial \eta} \dot{\eta} \dot{\xi} + \frac{\partial M_\eta}{\partial \eta} (\dot{\eta})^2 + \frac{\partial M_\eta}{\partial \xi} \dot{\xi} \dot{\eta}) \\ &\quad - e \left(\frac{\partial A_\eta}{\partial \xi} \dot{\xi} + \frac{\partial A_\eta}{\partial \eta} \dot{\eta} + \frac{\partial A_\eta}{\partial t} \right). \end{aligned}$$

We end up with the motions equations in this general coordinates system:

$$\begin{aligned} m(M_\xi \ddot{\xi} + M_{\xi\eta} \ddot{\eta}) &= e \left(\frac{\partial A_\xi}{\partial \xi} \dot{\xi} + \frac{\partial A_\xi}{\partial \eta} \dot{\eta} + \frac{\partial A_\xi}{\partial t} \right) \\ &\quad + m \left(\left(\frac{-1}{2} \frac{\partial M_\xi}{\partial \xi} \right) \xi^2 + \left(\frac{1}{2} \frac{\partial M_\eta}{\partial \xi} - \frac{\partial M_{\xi\eta}}{\partial \eta} \right) \eta^2 - \frac{\partial M_\xi}{\partial \eta} \dot{\xi} \dot{\eta} \right) \\ &\quad - e \left(\frac{\partial A_\xi}{\partial \xi} \dot{\xi} + \frac{\partial A_\eta}{\partial \xi} \dot{\eta} - \frac{\partial \phi}{\partial \xi} \right), \end{aligned}$$

$$\begin{aligned} m(M_\eta \ddot{\eta} + M_{\xi\eta} \ddot{\xi}) &= +e \left(\frac{\partial A_\eta}{\partial \xi} \dot{\xi} + \frac{\partial A_\eta}{\partial \eta} \dot{\eta} + \frac{\partial A_\eta}{\partial t} \right) \\ &\quad + m \left(\left(\frac{1}{2} \frac{\partial M_\xi}{\partial \eta} - \frac{\partial M_{\xi\eta}}{\partial \xi} \right) \xi^2 + \left(\frac{-1}{2} \frac{\partial M_\eta}{\partial \eta} \right) \eta^2 - \frac{\partial M_\eta}{\partial \xi} \dot{\xi} \dot{\eta} \right) \\ &\quad - e \left(\frac{\partial A_\xi}{\partial \eta} \dot{\xi} + \frac{\partial A_\eta}{\partial \eta} \dot{\eta} - \frac{\partial \phi}{\partial \eta} \right). \end{aligned}$$

But, we know that :

$$\mathbf{E} = -\frac{\partial \mathbf{A}}{\partial t} - \nabla \phi = \begin{pmatrix} -\frac{\partial A_\xi}{\partial t} - \frac{\partial \phi}{\partial \xi} \\ -\frac{\partial A_\eta}{\partial t} - \frac{\partial \phi}{\partial \eta} \end{pmatrix},$$

$$\dot{\mathbf{q}} \wedge \begin{pmatrix} \frac{B_\xi M_\eta - B_\eta M_{\xi\eta}}{\det(J)} \\ \frac{B_\eta M_\xi - B_\xi M_{\xi\eta}}{\det(J)} \end{pmatrix} = \dot{\mathbf{q}} \wedge (\nabla \wedge \mathbf{A}) = \begin{pmatrix} \dot{\eta} \left(\frac{\partial A_\eta}{\partial \xi} - \frac{\partial A_\xi}{\partial \eta} \right) \\ -\dot{\xi} \left(\frac{\partial A_\eta}{\partial \xi} - \frac{\partial A_\xi}{\partial \eta} \right) \end{pmatrix},$$

with (B_ξ, B_η) the coordinates of \mathbf{B} in the new system.

Therefore, we get,

C.2. General coordinates

$$\begin{aligned}
M_\xi \ddot{\xi} + M_{\xi\eta} \ddot{\eta} &= \left(\frac{-1}{2} \frac{\partial M_\xi}{\partial \xi} \right) \dot{\xi}^2 + \left(\frac{1}{2} \frac{\partial M_\eta}{\partial \xi} - \frac{\partial M_{\xi\eta}}{\partial \eta} \right) \dot{\eta}^2 - \frac{\partial M_\xi}{\partial \eta} \dot{\xi} \dot{\eta} \\
&- \frac{e}{m} (\mathbf{E} + \dot{\mathbf{q}} \wedge (\nabla \wedge \mathbf{A})) |_\xi, \\
M_\eta \ddot{\eta} + M_{\xi\eta} \ddot{\xi} &= \left(\frac{1}{2} \frac{\partial M_\xi}{\partial \eta} - \frac{\partial M_{\xi\eta}}{\partial \xi} \right) \dot{\xi}^2 + \left(\frac{-1}{2} \frac{\partial M_\eta}{\partial \eta} \right) \dot{\eta}^2 - \frac{\partial M_\eta}{\partial \xi} \dot{\xi} \dot{\eta} \\
&- \frac{e}{m} (\mathbf{E} + \dot{\mathbf{q}} \wedge (\nabla \wedge \mathbf{A})) |_\eta,
\end{aligned}$$

where derivatives with respect to ξ and η were given previously.

This is equivalent to the system:

$$\left\{ \begin{array}{l}
M_\xi \ddot{\xi} + M_{\xi\eta} \ddot{\eta} - \left(\left(\frac{-1}{2} \frac{\partial M_\xi}{\partial \xi} \right) \dot{\xi}^2 + \left(\frac{1}{2} \frac{\partial M_\eta}{\partial \xi} - \frac{\partial M_{\xi\eta}}{\partial \eta} \right) \dot{\eta}^2 - \frac{\partial M_\xi}{\partial \eta} \dot{\xi} \dot{\eta} \right) \\
= -\frac{e}{m} (\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_\xi \\
M_\eta \ddot{\eta} + M_{\xi\eta} \ddot{\xi} - \left(\left(\frac{1}{2} \frac{\partial M_\xi}{\partial \eta} - \frac{\partial M_{\xi\eta}}{\partial \xi} \right) \dot{\xi}^2 + \left(\frac{-1}{2} \frac{\partial M_\eta}{\partial \eta} \right) \dot{\eta}^2 - \frac{\partial M_\eta}{\partial \xi} \dot{\xi} \dot{\eta} \right) \\
= -\frac{e}{m} (\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_\eta
\end{array} \right.$$

where $\dot{\mathbf{q}} = (v_\xi, v_\eta) = (\dot{\xi}, \dot{\eta})$ are the component of the velocity in the new coordinates. In order to implement these equations, we need to compute explicitly the coefficients. We get after simplifications:

$$\left\{ \begin{array}{l}
\det(J) \frac{d\dot{\xi}}{dt} + \dot{\xi}^2 K_{\xi,\eta} + \dot{\eta}^2 K_{\eta,\eta} + 2 \dot{\eta} \dot{\xi} K_{\eta\xi,\eta} \\
= -\frac{e}{m \det(J)} ((\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_\xi) M_\eta - (\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_\eta) M_{\xi\eta} \\
\det(J) \frac{d\dot{\eta}}{dt} - \dot{\xi}^2 K_{\xi,\xi} - \dot{\eta}^2 K_{\eta,\xi} - 2 \dot{\eta} \dot{\xi} K_{\xi\eta,\xi} \\
= -\frac{e}{m \det(J)} ((\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_\eta) M_\xi - (\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_\xi) M_{\xi\eta}
\end{array} \right.$$

with

$$\begin{aligned}
K_{\xi,\xi} &= H_\xi V_\xi, \\
K_{\xi,\eta} &= H_\xi V_\eta, \\
K_{\eta,\eta} &= H_\eta V_\eta, \\
K_{\eta,\xi} &= H_\eta V_\xi, \\
K_{\xi\eta,\xi} &= H_{\xi\eta} V_\xi, \\
K_{\xi\eta,\eta} &= H_{\xi\eta} V_\eta,
\end{aligned}$$

C.3. Numerical implementation

where

$$\begin{aligned}
 H_\xi &= \begin{pmatrix} \frac{\partial^2 F_1}{\partial^2 \xi} \\ \frac{\partial^2 F_2}{\partial^2 \xi} \end{pmatrix}, \\
 H_\eta &= \begin{pmatrix} \frac{\partial^2 F_1}{\partial^2 \eta} \\ \frac{\partial^2 F_2}{\partial^2 \eta} \end{pmatrix}, \\
 H_{\xi\eta} &= \begin{pmatrix} \frac{\partial^2 F_1}{\partial \xi \partial \eta} \\ \frac{\partial^2 F_2}{\partial \xi \partial \eta} \end{pmatrix}, \\
 V_\xi &= \begin{pmatrix} \frac{\partial F_2}{\partial \xi} \\ -\frac{\partial F_1}{\partial \xi} \end{pmatrix}, \\
 V_\eta &= \begin{pmatrix} \frac{\partial F_2}{\partial \eta} \\ -\frac{\partial F_1}{\partial \eta} \end{pmatrix},
 \end{aligned}$$

and $\det(J) = \frac{\partial F_1}{\partial \xi} \frac{\partial F_2}{\partial \eta} - \frac{\partial F_1}{\partial \eta} \frac{\partial F_2}{\partial \xi}$ corresponds to the determinant of the change of variable Jacobian.

C.3 Numerical implementation

Equations

For $\dot{\theta} = 0$, we must solve,

$$\left\{ \begin{array}{l}
 M_\xi \ddot{\xi} + M_{\xi\eta} \ddot{\eta} - \left(\left(\frac{-1}{2} \frac{\partial M_\xi}{\partial \xi} \right) \xi^2 + \left(\frac{1}{2} \frac{\partial M_\eta}{\partial \xi} - \frac{\partial M_{\xi\eta}}{\partial \eta} \right) \dot{\eta}^2 - \frac{\partial M_\xi}{\partial \eta} \xi \dot{\eta} \right) \\
 = -\frac{e}{m} (\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_\xi \\
 \\
 M_\eta \ddot{\eta} + M_{\xi\eta} \ddot{\xi} - \left(\left(\frac{1}{2} \frac{\partial M_\xi}{\partial \eta} - \frac{\partial M_{\xi\eta}}{\partial \xi} \right) \xi^2 + \left(\frac{-1}{2} \frac{\partial M_\eta}{\partial \eta} \right) \dot{\eta}^2 - \frac{\partial M_\eta}{\partial \xi} \xi \dot{\eta} \right) \\
 = -\frac{e}{m} (\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_\eta
 \end{array} \right.$$

where $\dot{\mathbf{q}} = (v_\xi, v_\eta, v_\theta) = (\dot{\xi}, \dot{\eta}, \dot{\theta})$ are the components of the velocity in the new coordinates system. In order to implement these equations, we need to linearize them, and compute explicitly the coefficients. After simplifications, we get,

$$\left\{ \begin{array}{l}
 \det(J) \frac{d\dot{\xi}}{dt} + \dot{\xi}^2 K_{\xi,\eta} + \dot{\eta}^2 K_{\eta,\eta} + 2 \dot{\eta} \dot{\xi} K_{\eta\xi,\eta} \\
 = -\frac{e}{m \det(J)} ((\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_\xi) M_\eta - (\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_\eta) M_{\xi\eta} \\
 \\
 \det(J) \frac{d\dot{\eta}}{dt} - \dot{\xi}^2 K_{\xi,\xi} - \dot{\eta}^2 K_{\eta,\xi} - 2 \dot{\eta} \dot{\xi} K_{\xi\eta,\xi} \\
 = -\frac{e}{m \det(J)} ((\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_\eta) M_\xi - (\mathbf{E} + \dot{\mathbf{q}} \wedge \mathbf{B}) |_\xi) M_{\xi\eta}
 \end{array} \right.$$

C.4. Computing densities ρ and \mathbf{J}

with

$$\begin{aligned} K_{\xi,\xi} &= H_{\xi}V_{\xi}, \\ K_{\xi,\eta} &= H_{\xi}V_{\eta}, \\ K_{\eta,\eta} &= H_{\eta}V_{\eta}, \\ K_{\eta,\xi} &= H_{\eta}V_{\xi}, \\ K_{\xi\eta,\xi} &= H_{\xi\eta}V_{\xi}, \\ K_{\xi\eta,\eta} &= H_{\xi\eta}V_{\eta}, \end{aligned}$$

where

$$\begin{aligned} H_{\xi} &= \begin{pmatrix} \frac{\partial^2 F_1}{\partial^2 \xi} \\ \frac{\partial^2 F_2}{\partial^2 \xi} \end{pmatrix} \\ H_{\eta} &= \begin{pmatrix} \frac{\partial^2 F_1}{\partial^2 \eta} \\ \frac{\partial^2 F_2}{\partial^2 \eta} \end{pmatrix} \\ H_{\xi\eta} &= \begin{pmatrix} \frac{\partial^2 F_1}{\partial \xi \partial \eta} \\ \frac{\partial^2 F_2}{\partial \xi \partial \eta} \end{pmatrix} \\ V_{\xi} &= \begin{pmatrix} \frac{\partial F_2}{\partial \xi} \\ -\frac{\partial F_1}{\partial \xi} \end{pmatrix} \\ V_{\eta} &= \begin{pmatrix} \frac{\partial F_2}{\partial \eta} \\ -\frac{\partial F_1}{\partial \eta} \end{pmatrix}, \end{aligned}$$

and $\det(J) = \frac{\partial F_1}{\partial \xi} \frac{\partial F_2}{\partial \eta} - \frac{\partial F_1}{\partial \eta} \frac{\partial F_2}{\partial \xi}$ corresponds to the change of variable determinant.

C.4 Computing densities ρ and \mathbf{J}

In cartesian coordinates, charge and current densities ρ and \mathbf{J} are given by :

$$\begin{aligned} \rho(\mathbf{X}, t) &= - \int \int \int_{\mathbf{V}} f(\mathbf{X}, \mathbf{V}, t) d\mathbf{V}, \\ \mathbf{J}(\mathbf{X}, t) &= - \int \int \int_{\mathbf{V}} f(\mathbf{X}, \mathbf{V}, t) \mathbf{V} d\mathbf{V}. \end{aligned}$$

Using the expression of the density in term of Dirac's sum, we get:

$$\begin{aligned} \rho(\mathbf{X}, t) &= - \int \int \int_{\mathbf{V}} \sum_{k=1}^N \omega_k \delta(\mathbf{X} - \mathbf{X}_{\mathbf{k}}(t)) \delta(\mathbf{V} - \mathbf{V}_{\mathbf{k}}(t)) d\mathbf{V} \\ &= - \sum_{k=1}^N \omega_k \delta(\mathbf{X} - \mathbf{X}_{\mathbf{k}}(t)), \\ \mathbf{J}(\mathbf{X}, t) &= - \int \int \int_{\mathbf{V}} \sum_{k=1}^N \omega_k \delta(\mathbf{X} - \mathbf{X}_{\mathbf{k}}(t)) \delta(\mathbf{V} - \mathbf{V}_{\mathbf{k}}(t)) \mathbf{V} d\mathbf{V} \\ &= - \sum_{k=1}^N \omega_k \mathbf{V}_{\mathbf{k}}(t) \delta(\mathbf{X} - \mathbf{X}_{\mathbf{k}}(t)). \end{aligned}$$

C.4. Computing densities ρ and \mathbf{J}

In order to implement these expressions, we need to compute them on each cell. This can be done easily on the Patch domain, rather than the physical one. We need to use a change of variable.

Change of variables

We follow the same idea as previously. We first go to cylindrical coordinates:

$$\mathbf{X} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} r \cos \theta \\ r \sin \theta \\ z \end{pmatrix} = \begin{pmatrix} G_1(z, r, \theta) \\ G_2(z, r, \theta) \\ G_3(z, r, \theta) \end{pmatrix} = \mathbf{G}(z, r, \theta),$$

and then, go to a general coordinate system:

$$\begin{pmatrix} z \\ r \\ \theta \end{pmatrix} = \begin{pmatrix} F_1(\xi, \eta, \theta) \\ F_2(\xi, \eta, \theta) \\ F_3(\xi, \eta, \theta) \end{pmatrix} = \mathbf{F}(\xi, \eta, \theta),$$

and,

$$\mathbf{X} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} (GoF)_1(\xi, \eta, \theta) \\ (GoF)_2(\xi, \eta, \theta) \\ (GoF)_3(\xi, \eta, \theta) \end{pmatrix} = \mathbf{GoF}(\xi, \eta, \theta) = \mathbf{GoF}(\mathbf{U}).$$

We denote by Jac the jacobian of this change of coordinates.

Expression of ρ

Let C be a cell in the physical domain, and \tilde{C} be the correspondent cell in the patch ($C = \mathbf{F}(\tilde{C})$). We have for the charge density,

$$\begin{aligned} \rho_C(t) &= \int_C \rho(\mathbf{X}, t) d\mathbf{X} \\ &= \int_{\tilde{C}} \rho(\mathbf{GoF}(\mathbf{U}), t) |Jac| d\mathbf{U} \\ &= - \int_{\tilde{C}} \sum_{k=1}^N \omega_k \delta(\mathbf{GoF}(\mathbf{U}) - \mathbf{GoF}(\mathbf{U}_k(t))) |Jac| d\mathbf{U} \\ &= - \int_{\tilde{C}} \sum_{k=1}^N \omega_k \delta(\mathbf{U} - \mathbf{U}_k(t)) |Jac| d\mathbf{U} \\ &= - \sum_{k|\mathbf{U}_k(t) \in \tilde{C}} \omega_k |Jac|. \end{aligned}$$

In order to average, we divide this expression by the area of the cell,

$$\rho_C(t) = - \frac{1}{\Delta \mathbf{U}} \sum_{k|\mathbf{U}_k(t) \in \tilde{C}} \omega_k |Jac|.$$

Expression of \mathbf{J}

For the current density we have,

$$\begin{aligned}
 \mathbf{J}_C(t) &= \int_C \mathbf{J}(\mathbf{X}, t) d\mathbf{X} \\
 &= \int_{\tilde{C}} \mathbf{J}(\mathbf{G}\circ\mathbf{F}(\mathbf{U}), t) |Jac| d\mathbf{U} \\
 &= - \int_{\tilde{C}} \sum_{k=1}^N \omega_k \mathbf{V}_k(t) \delta(\mathbf{G}\circ\mathbf{F}(\mathbf{U}) - \mathbf{G}\circ\mathbf{F}(\mathbf{U}_k(t))) |Jac| d\mathbf{U} \\
 &= - \int_{\tilde{C}} \sum_{k=1}^N \omega_k \mathbf{V}_k(t) \delta(\mathbf{U} - \mathbf{U}_k(t)) |Jac| d\mathbf{U} \\
 &= - \sum_{k|\mathbf{U}_k(t) \in \tilde{C}} \omega_k \mathbf{V}_k(t) |Jac|.
 \end{aligned}$$

In order to average, we divide this expression by the area of the cell,

$$\mathbf{J}_C(t) = -\frac{1}{\Delta\mathbf{U}} \sum_{k|\mathbf{U}_k(t) \in \tilde{C}} \omega_k \mathbf{V}_k(t) |Jac|,$$

where \mathbf{V}_k is the velocity of the particle k in the physical domain.

Python Interface

Contents

D.1 Introduction	182
D.2 pdefield class	182
D.3 source class	182
D.4 isogeo class	182
D.4.1 Setting Grid's parameters	184
D.4.2 Poisson's equation	185
D.4.3 Anisotropic Diffusion	190
D.4.4 Maxwell's 2D problem	193
D.5 Using GB-splines	206
D.6 PyIGA's available operators	206
D.7 PyIGA's input data	208
D.7.1 Importing domain data	208
D.7.2 Refinement data	209
D.7.3 Boundary conditions	210
D.7.4 Silver Muller condition	211
D.7.5 Details and output file	211
D.8 Creating domains using the GUI	211
D.8.1 The XML Format	211
D.8.2 Using the GUI	211
D.8.3 Examples	212
D.9 Creating domains defined by an implicit function	213
D.10 Visualization	215
D.10.1 Using Silo	215
D.10.2 Using VTK	216
D.11 PyIGA and Pastix	216
D.12 Installing PyIGA	216

D.1. Introduction

D.1 Introduction

In this chapter we introduce the *PyIGA* library. We will also present some examples of solving partial differential equations using the *IGA* approach.

PyIGA is a library written in Fortran and Python. The main idea behind *PyIGA*, is to facilitate solving pde's; the user will only need to write a Python-script and then execute it. The book [79] is an excellent introduction to Python for Computational Science. The reader may find in it, everything he will need.

D.2 pdefield class

In *PyIGA*, each scalar or vectorial function unknown, will be modeled by what we call a *field*. The constructor of a *field* object needs the following parameters:

- **as_name** : the name used for visualization diagnostics,
- **ai_ndof** : the dimension (scalar or vectorial) of the field, **default value** : scalar function
- **ai_bc** : the type of the boundary condition, **default value** : 0.

when calling *iso.addfield*, this assigns an *id* to the created field.

```
F = pdefield ( 'L' )
iso.addfield ( F )
```

D.3 source class

We have implemented a class to handle source terms. For the moment, only L^2 projector operator is implemented. But in the future, we can implement quasi-interpolants (in the case of *GB-splines* for example).

A source term must be associated to a given field:

```
S = source ( F )
iso.addsource ( S )
```

D.4 isogeo class

In this section, we will show, step by step, how one can use *PyIGA* to solve a scalar pde. To this purpose we consider solving the Poisson's equation on a given domain $\Omega \subset \mathbb{R}^2$. This begins by importing the class *isogeo*, and create the associated object,

```
from isogeo import *
iso = isogeo ( )
```

now, we can choose the operators (matrices) involved in the pde,

D.4. isogeo class

```
# creation of the involved matrices
stiffness_id = iso.addmatrix( F, F, iso.STIFFNESS )
```

To finish the definition of the operator, we need to give the parameters of the matrix,

```
def parammatrices(work, values):
    #we must fill matrices params for all matrices
    #values[0,0:nparam] =
    #...
    #values[self.nmatrices, 0:nparam] =
    #####
    #first matrix
    values[1,0] = 1.0
    values[1,1] = 0.0
    values[1,2] = 0.0
    values[1,3] = 1.0
```

now, we initialize *isogeo*,

```
iso.initiso ( )
iso.initnurbsfem ( )
```

in the initialization step, *PyIGA* performs certain tasks. For example, this constructs the *Gauss-Quadrature* grid. All through this document, we will see other tasks that are performed by *PyIGA* (the construction of a DeRham sequence, using **H-div** or **H-rot** formulation). We now have to evaluate input functions on the grid.

```
#evaluating both sources and matrices params on the GL grid
iso.eval_functions_on_glgrid(af_source = sources, af_matrices = ←
    parammatrices)
```

Now, we can assemble the matrices involved in our pde, source terms and also the exact solutions if they are given,

```
iso.fem.assembly()
```

using, **iso.csr_matrix**, we can construct a **scipy.sparse** matrix using the *id* of the matrix,

```
from scipy.sparse import *
Stiffness = iso.csr_matrix ( stiffness_id )
```

to solve the linear system, we must export the source term,

```
li_dim = F.dim
lpr_source = numpy.zeros(li_dim , dtype=numpy.double)
lpr_source = iso.getsource ( S )
```

D.4. isogeo class

to solve the linear system, we can call `spsolve`,

```
from scipy.sparse.linalg import spsolve

lpr_tmp = numpy.zeros(li_dim , dtype=numpy.double)

lpr_tmp = spsolve ( Stiffness, lpr_source )
```

The user can also call *SuperLU*,

```
from scipy.sparse.linalg import splu

#----- factorizing matrices -----
Stiffness_csc = Stiffness.tocsc()
op_Stiffness = splu( Stiffness_csc )

#----- Solving the linear system -----
lpr_tmp = op_Stiffness.solve ( lpr_source )
```

For diagnostics, we need to import the solution into the `isogeo` module,

```
iso.setfield ( F , lpr_tmp )

#diagnostics
iso.silodiag(exact)
iso.l2errordiag(exact)
```

The argument function (here *exact*), can be omitted, if the user does not know it. In this case, *PyIGA* will write only the numerical solution. For more details about that, we refer to the section (D.10):

```
#diagnostics
iso.silodiag ( )
```

Remark D.4.1 For the moment, *PyIGA* handles only *sil* files, for visualization using *Visit*.

The user can export or import the matrix, in the Matrix Market format, by calling :

```
#--- exporting matrix
mmwrite('./Runs/Stiffness.mtx', Stiffness)

#--- importing matrix
Stiffness = mmread('./Runs/Stiffness.mtx')
```

D.4.1 Setting Grid's parameters

The *isogeo* class contains some functions that allow the user to call the *Python* script with arguments.

D.4. isogeo class

- **href** : the number of knot to insert (h-refinement) , assigned using `-href=n1,n2`, or `-n n1 n2`
- **pref** : the order used to elevate the *NURBS/B-splines* curves (p-refinement) , assigned using `-pref=p1,p2`, or `-p p1 p2`
- **ordergl** : the number of Quadratures points, `-g (-ordergl=val)` : this assigns the value val to ordergl
- **dt** : `-t (-dt=val)` : this assigns the value val to dt
- **niter** : `-i (-niter=val)` : this assigns the value val to niter
- **nfreq** : `-f (-nfreq=val)` : this assigns the value val to nfreq
- **load_matrices** : `-M (-loadm)` : this tells *PyIGA* to load matrices, rather than compute them again
- **load_vectors** : `-V (-loadv)` : this tells *PyIGA* to load vectors, rather than computed them again
- **noviz** : `-z (-noviz)` : if you do not want to do visualization diagnostics
- **noerror** : `-e (-noerror)` : if you do not want to compute L2-norm errors

D.4.2 Poisson's equation

On a square domain : `pdelaplace_square.py`

Code Listing D.1: Poisson's equation on a square domain

```
#!/usr/bin/env python
from isogeo import *
import numpy
from scipy.sparse import *
from scipy.sparse.linalg import spsolve
import sys
import os
from time import clock, time
from scipy.io import mmread, mmwrite
#from pastrix import *
from get_arg_isogeo import *

#####
# reading arguments must be done before calling set_data
# otherwise sys.argv will be erased
href = get_arg_href ( sys.argv [ 1 : ] )
pref = get_arg_pref ( sys.argv [ 1 : ] )
ordergl = get_arg_ordergl ( sys.argv [ 1 : ] )
#####

#####
# setting up domain data
sys.argv=["set_data.py","unit_square","-1"]

PYIGAPATH = os.getenv("PYIGAPATH")
setdatafile=os.path.join(PYIGAPATH,"set_data.py")
execfile(setdatafile)
#####
```

D.4. isogeo class

```
#####
L = 1.0
mode = 2
lamb = mode * pi / L

#####
#
#   test pour le carre
#
#####
def sources(work, values):
    #we must fill source term for all pdes
    #values[0] =
    #...
    #values[self.npdes] =
    values[0] = 2.0 * ( (lamb)**2 ) * sin(lamb*work[0]) * sin(lamb*work[1])

def parammatrices(work, values):
    #we must fill matrices params for all matrices
    #values[0,0:nparam] =
    #...
    #values[self.nmatrices, 0:nparam] =
    #####
    #first matrix
    values[0,0] = 1.0
    #second matrix
    values[1,0] = 1.0
    values[1,1] = 0.0
    values[1,2] = 0.0
    values[1,3] = 1.0

def exact(work, values):
    #we must fill source term for all pdes
    #values[0] =
    #...
    #values[self.npdes] =
    values[0] = sin(lamb*work[0]) * sin(lamb*work[1])
#####
iso = isogeo ( )

F = pdefield ( 'L' )
iso.addfield ( F )

S = source ( F )
iso.addsource ( S )

iso.fem.set_href ( href )
iso.fem.set_pref ( pref )
iso.fem.set_ordergl ( ordergl )

#... test params
#iso.fem.set_detail ( 3 )
iso.fem.set_boundarycondition ( 10 )
iso.fem.set_detail_silo ( 4 )
#...

# creation of the involved matrices
mass_id      = iso.addmatrix( F, F, iso.MASS )
stiffness_id = iso.addmatrix( F, F, iso.STIFNESS )

iso.initiso ( )

iso.initnurbsfem ( )

#evaluating both sources and matrices params on the GL grid
iso.eval_functions_on_glgrid(af_source = sources, af_matrices = parammatrices)
```

D.4. isogeo class

```
iso.fem.assembly()

Stiffness = iso.csr_matrix ( stiffness_id )
Mass      = iso.csr_matrix ( mass_id      )

mmwrite('../Runs/Stiffness.mtx', Stiffness)
mmwrite('../Runs/Mass.mtx', Mass)

#— using Pastix
#Stiff_past = pastrix ( pastix, fem, stiffness_id )
#—

li_dim = F.dim

lpr_source = numpy.zeros(li_dim ,dtype=numpy.double)
lpr_tmp     = numpy.zeros(li_dim ,dtype=numpy.double)

lpr_source = iso.getsource ( S )

#— utilisation de Spsolve
start = clock()

lpr_tmp = spsolve ( Stiffness, lpr_source )

elapsed = ( clock() - start )
print ( "SPSOLVE – CPU time for solving the linear system is :" + str ( elapsed ) )
#—

#— utilisation de Pastix
#start = clock()

#lpr_tmp = lpr_source
#Stiff_pastPsi.solve ( lpr_tmp )

#elapsed = ( clock() - start )
#print ( "PASTIX – CPU time for solving the linear system is :" + str ( elapsed ) )
#—

iso.setfield ( F , lpr_tmp )

#diagnostics
iso.silodiag(exact)
iso.l2errordiag(exact)

lr_normerror = iso.fem.getnormerror ( F.id, 1 )
print("error :"+str(lr_normerror))
lr_hmax = iso.fem.get_hmax ( )
print("hmax :"+str(lr_hmax))

lpr_output = []
lpr_output.append(lr_hmax)
lpr_output.append(lr_normerror)

lo_file = open('output.log', 'w')
lo_file.write(str(lpr_output))
lo_file.close()

iso.free ( )
```

On a circular domain : pdelaplace_circle.py

We only need to change the corresponding functions, for the source term and the exact solution,

D.4. isogeo class

Code Listing D.2: Poisson's equation on a circular domain

```
#!/usr/bin/env python
from isogeo import *
import numpy
from scipy.sparse import *
from scipy.sparse.linalg import spsolve
import sys
import os
from time import clock, time
from scipy.io import mmread, mmwrite
from get_arg_isogeo import *

#####
# reading arguments must be done before calling set_data
# otherwise sys.argv will be erased
href = get_arg_href ( sys.argv [ 1 : ] )
pref = get_arg_pref ( sys.argv [ 1 : ] )
ordergl = get_arg_ordergl ( sys.argv [ 1 : ] )
#####

#####
# setting up domain data
sys.argv=["set_data.py", "circle_1.0", "-1"]

PYIGAPATH = os.getenv("PYIGAPATH")
setdatafile=os.path.join(PYIGAPATH, "set_data.py")
execfile(setdatafile)
#####

#####

#####
#
#   test pour le cercle
#
#####
def sources(work, values):
    #we must fill source term for all pdes
    #values[0] =
    #...
    #values[self.npdes] =
    r2 = work[0]**2 + work[1]**2
    w = 1.0 - r2
    values[0] = 4.0 * r2 * sin ( w ) + 4.0 * cos ( w )

def parammatrices(work, values):
    #we must fill matrices params for all matrices
    #values[0,0:nparam] =
    #...
    #values[self.nmatrices, 0:nparam] =
    #mass matrix
    values[0,0] = 1.0
    #stiffness matrix
    values[1,0] = 1.0
    values[1,1] = 0.0
    values[1,2] = 0.0
    values[1,3] = 1.0

def exact(work, values):
    #we must fill source term for all pdes
    #values[0] =
    #...
    #values[self.npdes] =
    r2 = work[0]**2 + work[1]**2
    values[0] = sin ( 1.0 - r2 )
#####
iso = isogeo ( )
```


D.4. isogeo class

```
F = pdefield ( 'L' )
iso.addfield ( F )

S = source ( F )
iso.addsource ( S )

iso.fem.set_href ( href )
iso.fem.set_pref ( pref )
iso.fem.set_ordergl ( ordergl )

#... test params
#iso.fem.set_detail ( 3 )
iso.fem.set_boundarycondition ( 10 )
iso.fem.set_detail_silo ( 4 )
#iso.fem.set_extdomain(1)
#...

# creation of the involved matrices
mass_id      = iso.addmatrix( F, F, iso.MASS )
stiffness_id = iso.addmatrix( F, F, iso.STIFFNESS )

iso.initiso ( )

iso.initnurbsfem ( )

#evaluating both sources and matrices params on the GL grid
iso.eval_functions_on_glgrid(af_source = sources, af_matrices = parammatrices)

iso.fem.assembly()

Stiffness = iso.csr_matrix ( stiffness_id )
Mass      = iso.csr_matrix ( mass_id      )

mmwrite('../Runs/Stiffness.mtx', Stiffness)
mmwrite('../Runs/Mass.mtx', Mass)

li_dim = F.dim

lpr_source = numpy.zeros(li_dim ,dtype=numpy.double)
lpr_tmp    = numpy.zeros(li_dim ,dtype=numpy.double)

lpr_source = iso.getsource ( S )

#—— utilisation de Spsolve
start = clock()

lpr_tmp = spsolve ( Stiffness, lpr_source )

elapsed = ( clock() - start )
print ( "SPSOLVE – CPU time for solving the linear system is :" + str ( elapsed ) )
#——

iso.setfield ( F , lpr_tmp )

#diagnostics
iso.silodiag(exact)
iso.l2errordiag(exact)

lr_normerror = iso.fem.getnormerror ( F.id, 1 )
print("error :"+str(lr_normerror))
lr_hmax = iso.fem.get_hmax ( )
print("hmax :"+str(lr_hmax))

lpr_output = []
lpr_output.append(lr_hmax)
lpr_output.append(lr_normerror)
```

D.4. isogeo class

```
lo_file = open('output.log', 'w')
lo_file.write(str(lpr_output))
lo_file.close()

iso.free ( )
```

D.4.3 Anisotropic Diffusion

Code Listing D.3: Anisotropic Diffusion on a square domain

```
#!/usr/bin/env python
from isogeo import *
import numpy
import math
from scipy.sparse import *
from scipy.sparse.linalg import spsolve, splu
import sys
import os
from time import clock, time
from scipy.io import mmread, mmwrite
#from pastrix import *
from get_arg_isogeo import *

#####
# reading arguments must be done before calling set_data
# otherwise sys.argv will be erased
href = get_arg_href ( sys.argv [ 1 : ] )
pref = get_arg_pref ( sys.argv [ 1 : ] )
ordergl = get_arg_ordergl ( sys.argv [ 1 : ] )
dt = get_arg_dt ( sys.argv [ 1 : ] )
niter = get_arg_niter ( sys.argv [ 1 : ] )
load_matrices = get_arg_load_matrices ( sys.argv [ 1 : ] )
load_vectors = get_arg_load_vectors ( sys.argv [ 1 : ] )
noviz = get_arg_noviz ( sys.argv [ 1 : ] )
#####

#####
# setting up domain data
sys.argv=["set_data.py", "unit_square", "-1"]

PYIGAPATH = os.getenv("PYIGAPATH")
setdatafile=os.path.join(PYIGAPATH, "set_data.py")
execfile(setdatafile)
#####

#niter = 10000
nfreq = 500

t = 0.0
#dt = 0.0001

m1 = 0.25
m2 = 0.5
s1 = 0.001
s2 = 0.001

mu_paral = 2.0
mu_perp = 0.0
D = 0.00775

L1 = 0.025

def f(x,m,s):
    return exp(-(x-m)**2/(2.0*s))
```

D.4. isogeo class

```
def get_tetha(x,y):
    if ( x==0.0 ):
        if( y==0.0) :
            return 0.0

    lr_sign = 1.0
    if ( y < 0.0 ):
        lr_sign = -1.0

    lr_val = x / sqrt ( x**2 + y**2 )

    lr_result = lr_sign * acos ( lr_val )

    return lr_result
#####
#
#   test pour le cercle
#
#####
def sources(work, values):
    #we must fill source term for all pdes
    #values[0] =
    #...
    #values[self.npdes] =
    values[0] = f(work[0], m1,s1)*f(work[1], m2,s2)

def parammatrices(work, values):
    #we must fill matrices params for all matrices
    #values[0,0:nparam] =
    #...
    #values[self.nmatrices, 0:nparam] =
    #stiffness matrix
    lr_tetha = get_tetha ( work[0] , work[1] )

    lr_c = sin ( lr_tetha )
    lr_s = - cos ( lr_tetha )

    lr_c2 = lr_c**2
    lr_s2 = lr_s**2
    lr_sc = lr_s * lr_c

    values[0,0] = D * mu_parallel * lr_c2 + mu_perp * lr_s2
    values[0,1] = D * ( mu_parallel - mu_perp ) * lr_sc
    values[0,2] = D * ( mu_parallel - mu_perp ) * lr_sc
    values[0,3] = D * mu_parallel * lr_s2 + mu_perp * lr_c2
    #mass matrix
    values[1,0] = 1.0

def exact(work, values):
    #we must fill source term for all pdes
    #values[0] =
    #...
    #values[self.npdes] =
    values[0] = 0.0
#####
iso = isogeo ( )

F = pdefield ( 'L' )
iso.addfield ( F )

S = source ( F )
iso.addsource ( S )

iso.fem.set_href ( href )
iso.fem.set_pref ( pref )
iso.fem.set_ordergl ( ordergl )

#... test params
#iso.fem.set_detail ( 3 )
```

D.4. isogeo class

```
iso.fem.set_boundarycondition ( 10 )
iso.fem.set_detail_silo ( 4 )
#...

# creation of the involved matrices
stiffness_id = iso.addmatrix( F, F, iso.STIFFNESS )
mass_id      = iso.addmatrix( F, F, iso.MASS )

iso.initiso ( )

iso.initnurbsfem ( )

#evaluating both sources and matrices params on the GL grid
iso.eval_functions_on_glgrid(af_source = sources, af_matrices = parammatrices)

iso.fem.assembly()

Stiffness = iso.csr_matrix ( stiffness_id )
Mass      = iso.csr_matrix ( mass_id      )

mmwrite('../Runs/Stiffness.mtx', Stiffness)
mmwrite('../Runs/Mass.mtx', Mass)

#A contains mass+Stiffness
A = csr_matrix (Stiffness)
A = A + Mass

Mass_csc = Mass.tocsc()
op_Mass = splu( Mass_csc )

li_dim = F.dim

lpr_source = numpy.zeros(li_dim ,dtype=numpy.double)
lpr_tmpU   = numpy.zeros(li_dim ,dtype=numpy.double)
lpr_U      = numpy.zeros(li_dim ,dtype=numpy.double)
lpr_exact  = numpy.zeros(li_dim ,dtype=numpy.double)

#=====
lpr_source = iso.getsource ( S )

lpr_U = spsolve ( A , lpr_source )

iso.setfield ( S , lpr_U )

#diagnostics
iso.silodiag(exact,ai_numdiag=0)
iso.l2errordiag(exact)
#=====

nstart = 1
nend = niter + 1

def solve ( ai_nstart, ai_nend ):
    global t, dt, fem, iso
    global lpr_tmpU, lpr_dotU, lpr_U

    for i in range( ai_nstart, ai_nend ):
        print("===== iteration : "+str(i)          + " ←
              =====")
        t = t + dt

        lpr_tmpU = - A.dot ( lpr_U )

        lpr_dotU = op_Mass.solve ( lpr_tmpU )

        lpr_U = lpr_U + dt * lpr_dotU
        #print("lpr_U="+str(lpr_U))
```

D.4. isogo class

```
if ( mod ( i , nfreq ) == 0 ):
    numdiag = i / nfreq
    print("===                               diagnostic: "+str(numdiag)+" ←→
                                     ===")
    #print←→
    ("=====")←→

    print("iteration =" +str(i))
    numdiag = i / nfreq

    iso.setfield ( S , lpr_U )

    #diagnostics
    iso.silodiag(ai_numdiag=numdiag)

solve (nstart,nend)
print("to add more iterations , type the number you want")
n = input("press 0 to stop : ")
while ( n > 0 ):
    nstart = nend + 1
    nend += n
    solve (nstart,nend)
    print("to add more iterations , type the number you want")
    n = input("press 0 to stop : ")

iso.free()
```

D.4.4 Maxwell's 2D problem

In this section we show how we can solve Maxwell's 2D problem, using a **H-div** formulation. The key point here, is the use of a discrete DeRham sequence. We begin by the creation of the electrical and magnetic fields.

```
#creation of the magnetic field
B = pdefield ( 'B' )
iso.addfield ( B )

#creation of the electrical field
E = pdefield ( 'E', ai_ndof = 2 )
iso.addfield ( E )
```

The construction of the discrete DeRham sequence is done simply by calling,

```
#creation of the hilbert complex
iso.add_hilbertcomplex ( [B,E] , iso.HILBERT_COMPLEX_VW )
```

the default construction is **H-div** formulation. For **H-rot** formulation, we shall call,

```
#creation of the hilbert complex
iso.add_hilbertcomplex ( [B,E] , iso.HILBERT_COMPLEX_VW, ai_useHrot ←→
    True )
```

D.4. isogeo class

however, in this case we must take appropriate boundary conditions.
The user can also create the whole DeRham sequence:

```
#creation of the hilbert complex
iso.add_hilbertcomplex ( [B,E,Phi] , iso.HILBERT_COMPLEX_VWX )
```

We need to create also the source terms,

```
#creation of the sources terms
S_B = source ( B )
iso.addsource ( S_B )

S_E = source ( E )
iso.addsource ( S_E )
```

As it was shown before, we need to choose which operators (matrices) are involved in the two pdes, and then associated them to the corresponding pde.

```
# creation of the involved matrices
# Mass matrix
MassB_id      = iso.addmatrix ( B, B, iso.MASS )
# Mass matrix
MassE_id      = iso.addmatrix ( E, E, iso.MASS )
# Rotational matrix
rotational_id = iso.addmatrix ( B, E, iso.ROTATIONAL )
# R-matrix
rmatrix_id    = iso.addmatrix ( E, B, iso.R_MATRIX )
```

We now can initialize the **nurbsfem** module, evaluate sources terms, exact solutions and operators (matrices) parameters:

```
iso.initiso ( )
iso.initnurbsfem ( )

#evaluating both sources and matrices params on the GL grid
iso.eval_functions_on_glgrid(af_source = sources, af_matrices = ←
    parammatrices)

iso.fem.assembly()
```

using, **iso.csr_matrix**, we can construct a **scipy.sparse** matrix using the *id* of the matrix,

```
MassB      = iso.csr_matrix ( MassB_id      )
MassE      = iso.csr_matrix ( MassE_id      )
Rotational = iso.csr_matrix ( rotational_id )
Rmatrix    = iso.csr_matrix ( rmatrix_id    )
```

D.4. isogeo class

We give in the next sections the complete code for solving the Maxwell's 2D equations.

On a square domain :

Code Listing D.4: Maxwell's equations on a square domain

```
#!/usr/bin/env python
from isogeo import *
import numpy
from scipy.sparse import *
from scipy.sparse.linalg import spsolve, splu
import sys
import os
from time import clock, time
from scipy.io import mmread, mmwrite
from get_arg_isogeo import *

#####
# reading arguments must be done before calling set_data
# otherwise sys.argv will be erased
href = get_arg_href ( sys.argv [ 1 : ] )
pref = get_arg_pref ( sys.argv [ 1 : ] )
ordergl = get_arg_ordergl ( sys.argv [ 1 : ] )
dt = get_arg_dt ( sys.argv [ 1 : ] )
niter = get_arg_niter ( sys.argv [ 1 : ] )
load_matrices = get_arg_load_matrices ( sys.argv [ 1 : ] )
load_vectors = get_arg_load_vectors ( sys.argv [ 1 : ] )
noviz = get_arg_noviz ( sys.argv [ 1 : ] )
#####

#####
# setting up domain data
sys.argv=["set_data.py", "unit_square", "-1"]

PYIGAPATH = os.getenv("PYIGAPATH")
setdatafile=os.path.join(PYIGAPATH, "set_data.py")
execfile(setdatafile)
#####

#####
#
# GLOBAL DECLARATIONS
#
#####
load_data = load_matrices and load_vectors
export_matrices = True
export_vectors = True
nfreq = niter
timescheme =4

t = 0.0

L = 1.0
#####

#####
#
# test on a square domain
#
#####
mode1 = 1
mode2 = 1
```

D.4. isogeo class

```
k1 = model * pi / L
k2 = mode2 * pi / L

w = sqrt ( (k1)**2 + (k2)**2 )

def sources(work, values):
    #we must fill source term for all pdes
    #values[0] =
    #...
    #values[self.npdes] =
    #there is no source term
    values[0,0] = cos(k1*work[0]) \
        * cos(k2*work[1]) \
        * cos ( w * t )
    #electrical field, 1st component
    values[1,0] = ( - k2 / w ) * cos(k1*work[0]) \
        * sin(k2*work[1]) \
        * sin ( w * ( t - 0.5 * dt ) )
    #electrical field, 2nd component
    values[1,1] = ( k1 / w ) * sin(k1*work[0]) \
        * cos(k2*work[1]) \
        * sin ( w * ( t - 0.5 * dt ) )

def exact(work, values):
    #we must fill source term for all pdes
    #values[0] =
    #...
    #values[self.npdes] =
    #magnetic field

    values[0,0] = cos(k1*work[0]) \
        * cos(k2*work[1]) \
        * cos ( w * t )
    #electrical field, 1st component
    values[1,0] = ( - k2 / w ) * cos(k1*work[0]) \
        * sin(k2*work[1]) \
        * sin ( w * ( t - 0.5 * dt ) )
    #electrical field, 2nd component
    values[1,1] = ( k1 / w ) * sin(k1*work[0]) \
        * cos(k2*work[1]) \
        * sin ( w * ( t - 0.5 * dt ) )

def parammatrices(work, values):
    #we must fill matrices params for all matrices
    #values[0,0:nparam] =
    #...
    #values[self.nmatrices, 0:nparam] =
    #####
    #first matrix
    values[0,0] = 1.0
    values[0,1] = 1.0
    values[0,2] = 1.0
    values[0,3] = 1.0
    #second matrix
    values[1,0] = 1.0
    values[1,1] = 1.0
    values[1,2] = 1.0
    values[1,3] = 1.0
    #####

iso = isogeo ( )

#creation of the magnetic field
B = pdefield ( 'B' )
iso.addfield ( B )

#creation of the electrical field
E = pdefield ( 'E', ai_ndof = 2 )
iso.addfield ( E )
```


D.4. isogeo class

```
#creation of the hilbert complex
iso.add_hilbertcomplex ( [B,E] , iso.HILBERT_COMPLEX_VW )

#creation of the sources terms
S_B = source ( B )
iso.addsource ( S_B )

S_E = source ( E )
iso.addsource ( S_E )

iso.fem.set_href ( href )
iso.fem.set_pref ( pref )
iso.fem.set_ordergl ( ordergl )

#... test params
iso.fem.set_stdoutoutput ( 1 )
iso.fem.set_detail ( 1 )
iso.fem.set_boundarycondition ( 0 )
iso.fem.set_detail_silo ( 4 )
iso.useUnitPatch = True
#...

# creation of the involved matrices
# Mass matrix
MassB_id = iso.addmatrix ( B, B, iso.MASS )
# Mass matrix
MassE_id = iso.addmatrix ( E, E, iso.MASS )
# Rotational matrix
rotational_id = iso.addmatrix ( B, E, iso.ROTATIONAL )
# R-matrix
rmatrix_id = iso.addmatrix ( E, B, iso.R_MATRIX )

iso.initiso ( )
iso.initnurbsfem ( )

#evaluating both sources and matrices params on the GL grid
iso.eval_functions_on_glgrid(af_source = sources, af_matrices = parammatrices)

li_dimB = B.dim
li_dimE = E.dim

lpr_sourceE = numpy.zeros(li_dimE ,dtype=numpy.double)
lpr_E = numpy.zeros(li_dimE ,dtype=numpy.double)
lpr_dotE = numpy.zeros(li_dimE ,dtype=numpy.double)
lpr_tmpE = numpy.zeros(li_dimE ,dtype=numpy.double)

lpr_sourceB = numpy.zeros(li_dimB ,dtype=numpy.double)
lpr_B = numpy.zeros(li_dimB ,dtype=numpy.double)
lpr_dotB = numpy.zeros(li_dimB ,dtype=numpy.double)
lpr_tmpB = numpy.zeros(li_dimB ,dtype=numpy.double)

lpi_matrices = (MassB_id,MassE_id,rotational_id,rmatrix_id)
lpi_sources = (S_B.id ,S_E.id )

if ( load_matrices ) :
    MassB = mmread( '../Runs/MassB.mtx' )
    MassE = mmread( '../Runs/MassE.mtx' )
    Rotational = mmread( '../Runs/Rotational.mtx' )
    Rmatrix = mmread( '../Runs/Rmatrix.mtx' )

    lpi_matrices = ( )

if ( load_vectors ) :
    lpr_sourceB = genfromtxt ( "../Runs/sourceB.txt" )
```

D.4. isogo class

```
lpr_sourceE = genfromtxt ( "../Runs/sourceE.txt" )

lpi_sources = ()

if ( not load_data ) :

    iso.fem.matrixtoassembly ( lpi_matrices )
    iso.fem.sourcetoassembly ( lpi_sources )

    iso.fem.assembly()

    if ( not load_matrices ) :
        MassB      = iso.csr_matrix ( MassB_id      )
        MassE      = iso.csr_matrix ( MassE_id      )
        Rotational = iso.csr_matrix ( rotational_id )
        Rmatrix    = iso.csr_matrix ( rmatrix_id    )

    if ( not load_vectors ) :
        #computing the projection of the fields over the discrete spaces
        lpr_sourceB = iso.getsource ( S_B )
        lpr_sourceE = iso.getsource ( S_E )

if ( export_matrices ) :
    #———— exporting matrices —————
    mmwrite('../Runs/MassB.mtx', MassB)
    mmwrite('../Runs/MassE.mtx', MassE)
    mmwrite('../Runs/Rotational.mtx', Rotational)
    mmwrite('../Runs/Rmatrix.mtx', Rmatrix)
    #————

if ( export_vectors ) :
    #———— exporting vectors —————
    savetxt("../Runs/sourceB.txt", lpr_sourceB )
    savetxt("../Runs/sourceE.txt", lpr_sourceE )
    #————

#———— factorizing matrices —————
MassE_csc = MassE.tocsc()
op_MassE = splu( MassE_csc )

MassB_csc = MassB.tocsc()
op_MassB = splu( MassB_csc )
#————

lpr_B = op_MassB.solve ( lpr_sourceB )
lpr_E = op_MassE.solve ( lpr_sourceE )

iso.setfield ( B , lpr_B )
iso.setfield ( E , lpr_E )

#diagnostics
if ( not noviz ) :
    iso.silodiag ( exact )

iso.l2errordiag ( exact )

lr_hmax = iso.fem.get_hmax ( )
lr_normerrorB = iso.fem.getnormerror ( B.id, 1 )
lr_normerrorEx = iso.fem.getnormerror ( E.id, 1 )
lr_normerrorEy = iso.fem.getnormerror ( E.id, 2 )

lpr_output = []
lpr_output.append([lr_hmax,lr_normerrorB,lr_normerrorEx,lr_normerrorEy])
ls_file = "error_proj.txt"
savetxt( ls_file , lpr_output )

for i in range(1,niter+1):
```

D.4. isogeo class

```
t = t + dt

if ( timescheme == 2 ) :

    lpr_dotE = Rmatrix.dot ( lpr_B )

    lpr_E = lpr_E + dt * lpr_dotE

    lpr_tmpB = Rotational.dot ( lpr_E )

    lpr_dotB = op_MassB.solve ( lpr_tmpB )

    lpr_B = lpr_B + dt * lpr_dotB

if ( timescheme == 4 ) :

    #-----
    #      Magnetic field evaluation
    #-----
    lpr_dotE      = numpy.zeros(li_dimE ,dtype=numpy.double)
    lpr_tmp1_E    = numpy.zeros(li_dimE ,dtype=numpy.double)
    lpr_tmp2_E    = numpy.zeros(li_dimE ,dtype=numpy.double)

    lpr_dotB      = numpy.zeros(li_dimB ,dtype=numpy.double)
    lpr_tmp1_B    = numpy.zeros(li_dimB ,dtype=numpy.double)
    lpr_tmp2_B    = numpy.zeros(li_dimB ,dtype=numpy.double)
    lpr_tmp3_B    = numpy.zeros(li_dimB ,dtype=numpy.double)

    lpr_tmp1_E = Rmatrix.dot ( lpr_B )

    lpr_tmp1_B = Rotational.dot ( lpr_tmp1_E )

    lpr_tmp2_B = op_MassB.solve ( lpr_tmp1_B )

    lpr_tmp2_E = Rmatrix.dot ( lpr_tmp2_B )

    lpr_dot_E = lpr_tmp1_E + ( 1.0 / 24.0 ) * ( dt **2 ) * lpr_tmp2_E

    lpr_E = lpr_E + dt * lpr_dot_E

    #-----
    #      Magnetic field evaluation
    #-----
    lpr_dotE      = numpy.zeros(li_dimE ,dtype=numpy.double)
    lpr_tmp1_E    = numpy.zeros(li_dimE ,dtype=numpy.double)
    lpr_tmp2_E    = numpy.zeros(li_dimE ,dtype=numpy.double)

    lpr_dotB      = numpy.zeros(li_dimB ,dtype=numpy.double)
    lpr_tmp1_B    = numpy.zeros(li_dimB ,dtype=numpy.double)
    lpr_tmp2_B    = numpy.zeros(li_dimB ,dtype=numpy.double)
    lpr_tmp3_B    = numpy.zeros(li_dimB ,dtype=numpy.double)

    lpr_tmp1_B = Rotational.dot ( lpr_E )

    lpr_tmp2_B = op_MassB.solve ( lpr_tmp1_B )

    lpr_tmp1_E = Rmatrix.dot ( lpr_tmp2_B )

    lpr_tmp3_B = Rotational.dot ( lpr_tmp1_E )

    lpr_tmp1_B = op_MassB.solve ( lpr_tmp3_B )

    lpr_dot_B = lpr_tmp2_B + ( 1.0 / 24.0 ) * ( dt **2 ) * lpr_tmp1_B

    lpr_B = lpr_B + dt * lpr_dot_B

if ( mod ( i , nfreq ) == 0 ):
    print("iteration =" +str(i))
    numdiag = i / nfreq
```

D.4. isogeo class

```
iso.setfield ( B , lpr_B )
iso.setfield ( E , lpr_E )

#diagnostics
if ( not noviz ) :
    iso.silodiag ( exact , numdiag )

iso.l2errordiag ( exact )

lr_normerrorB = iso.fem.getnormerror ( B.id, 1 )
lr_normerrorEx = iso.fem.getnormerror ( E.id, 1 )
lr_normerrorEy = iso.fem.getnormerror ( E.id, 2 )

lpr_output = []
lpr_output.append([lr_hmax,lr_normerrorB,lr_normerrorEx,lr_normerrorEy])
ls_file = "error"+"_"+str(numdiag)+".txt"
savetxt( ls_file , lpr_output )

iso.free ( )
```

On a ring domain :

We need to use the **special** module of **scipy** for Bessel functions. We need also to change the corresponding functions, for the source term and the exact solution,

Code Listing D.5: Maxwell's equations on a ring domain

```
#!/usr/bin/env python
from isogeo import *
import numpy
from scipy.sparse import *
from scipy.sparse.linalg import spsolve, splu
from scipy import special as sp
from math import *
import sys
import os
from time import clock, time
from scipy.io import mmread, mmwrite
from get_arg_isogeo import *

#####
# reading arguments must be done before calling set_data
# otherwise sys.argv will be erased
href = get_arg_href ( sys.argv [ 1 : ] )
pref = get_arg_pref ( sys.argv [ 1 : ] )
ordergl = get_arg_ordergl ( sys.argv [ 1 : ] )
dt = get_arg_dt ( sys.argv [ 1 : ] )
niter = get_arg_niter ( sys.argv [ 1 : ] )
load_matrices = get_arg_load_matrices ( sys.argv [ 1 : ] )
load_vectors = get_arg_load_vectors ( sys.argv [ 1 : ] )
noviz = get_arg_noviz ( sys.argv [ 1 : ] )
#####

#####
# setting up domain data
sys.argv=["set_data.py","maxwell-ring","-1"]

PYIGAPATH = os.getenv("PYIGAPATH")
setdatafile=os.path.join(PYIGAPATH,"set_data.py")
execfile(setdatafile)
#####
#####
```

D.4. isogeo class

```
#
# GLOBAL DECLARATIONS
#
#####
load_data = load_matrices and load_vectors
export_matrices = True
export_vectors = True
nfreq = niter
timescheme =4

t = 0.0

r1 = 0.65138750344695903414
r2 = 0.99000418530735846839

rmin = 0.65138750344695903414
rmax = 0.99000418530735846839

a = 1.0
w = 3.0 * pi

#####

#####
#
# test pour la couronne
#
#####
def get_tetha(x,y):
    if ( x==0.0 ):
        if( y==0.0) :
            return 0.0

    lr_sign = 1.0
    if ( y < 0.0 ):
        lr_sign = -1.0

    lr_val = x / sqrt ( x**2 + y**2 )

    lr_result = lr_sign * acos ( lr_val )

    return lr_result

def sources(work, values):
    #we must fill source term for all pdes
    #values[0] =
    #...
    #values[self.npdes] =
    #there is no source term

    lr_radius = sqrt ( work[0]**2 + work[1]**2 )
    lr_tetha = get_tetha ( work[0] , work[1] )

    lr_J0 = sp.j0 ( w * lr_radius )
    lr_J1 = sp.j1 ( w * lr_radius )
    lr_Y0 = sp.y0 ( w * lr_radius )
    lr_Y1 = sp.y1 ( w * lr_radius )

    lr_J2 = 2.0 / ( w * lr_radius ) * lr_J1 - lr_J0
    lr_Y2 = 2.0 / ( w * lr_radius ) * lr_Y1 - lr_Y0

    values[0,0] = - cos ( w * t + lr_tetha ) \
        * ( lr_J1 + a * lr_Y1 )
    #electrical field, 1st component
    values[1,0] = - 0.5 * sin ( w * ( t - 0.5 * dt ) + lr_tetha ) * sin ( lr_tetha )←
        \
            * ( lr_J0 - lr_J2 + a * ( lr_Y0 - lr_Y2 ) ) \
            - cos ( lr_tetha ) / ( w * lr_radius ) \
            * cos ( w * ( t - 0.5 * dt ) + lr_tetha ) \
```

D.4. isogeo class

```

        * ( lr_J1 + a * lr_Y1 )

#electrical field , 2nd component
values[1,1] = 0.5 * sin ( w * ( t - 0.5 * dt ) + lr_tetha ) * cos ( lr_tetha ) \
        * ( lr_J0 - lr_J2 + a * ( lr_Y0 - lr_Y2 ) ) \
        - sin ( lr_tetha ) / ( w * lr_radius ) \
        * cos ( w * ( t - 0.5 * dt ) + lr_tetha ) \
        * ( lr_J1 + a * lr_Y1 )

def exact(work, values):
#we must fill source term for all pdes
#values[0] =
#...
#values[self.npdes] =
#magnetic field

lr_radius = sqrt ( work[0]**2 + work[1]**2 )
lr_tetha = get_tetha ( work[0] , work[1] )

lr_J0 = sp.j0 ( w * lr_radius )
lr_J1 = sp.j1 ( w * lr_radius )
lr_Y0 = sp.y0 ( w * lr_radius )
lr_Y1 = sp.y1 ( w * lr_radius )

lr_J2 = 2.0 / ( w * lr_radius ) * lr_J1 - lr_J0
lr_Y2 = 2.0 / ( w * lr_radius ) * lr_Y1 - lr_Y0

values[0,0] = - cos ( w * t + lr_tetha ) \
        * ( lr_J1 + a * lr_Y1 )
#electrical field , 1st component
values[1,0] = - 0.5 * sin ( w * ( t - 0.5 * dt ) + lr_tetha ) * sin ( lr_tetha ) ←
        \
        * ( lr_J0 - lr_J2 + a * ( lr_Y0 - lr_Y2 ) ) \
        - cos ( lr_tetha ) / ( w * lr_radius ) \
        * cos ( w * ( t - 0.5 * dt ) + lr_tetha ) \
        * ( lr_J1 + a * lr_Y1 )

#electrical field , 2nd component
values[1,1] = 0.5 * sin ( w * ( t - 0.5 * dt ) + lr_tetha ) * cos ( lr_tetha ) \
        * ( lr_J0 - lr_J2 + a * ( lr_Y0 - lr_Y2 ) ) \
        - sin ( lr_tetha ) / ( w * lr_radius ) \
        * cos ( w * ( t - 0.5 * dt ) + lr_tetha ) \
        * ( lr_J1 + a * lr_Y1 )

def parammatrices(work, values):
#we must fill matrices params for all matrices
#values[0,0:nparam] =
#...
#values[self.nmatrices, 0:nparam] =
#####
#first matrix
values[0,0] = 1.0
values[0,1] = 1.0
values[0,2] = 1.0
values[0,3] = 1.0
#second matrix
values[1,0] = 1.0
values[1,1] = 1.0
values[1,2] = 1.0
values[1,3] = 1.0
#####

iso = isogeo ( )

#creation of the magnetic field
B = pdefield ( 'B' )
iso.addfield ( B )

#creation of the electrical field

```

D.4. isogo class

```
E = pdefield ( 'E', ai_ndof = 2 )
iso.addfield ( E )

#creation of the hilbert complex
iso.add_hilbertcomplex ( [B,E] , iso.HILBERT_COMPLEX_VW )

#creation of the sources terms
S_B = source ( B )
iso.addsource ( S_B )

S_E = source ( E )
iso.addsource ( S_E )

iso.fem.set_href ( href )
iso.fem.set_pref ( pref )
iso.fem.set_ordergl ( ordergl )

#... test params
iso.fem.set_detail ( 1 )
iso.fem.set_boundarycondition ( 1 )
iso.fem.set_detail_silo ( 4 )
#...

# creation of the involved matrices
# Mass matrix
MassB_id = iso.addmatrix ( B, B, iso.MASS )
# Mass matrix
MassE_id = iso.addmatrix ( E, E, iso.MASS )
# Rotational matrix
rotational_id = iso.addmatrix ( B, E, iso.ROTATIONAL )
# R-matrix
rmatrix_id = iso.addmatrix ( E, B, iso.R_MATRIX )

iso.initiso ( )
iso.initnurbsfem ( )

#evaluating both sources and matrices params on the GL grid
iso.eval_functions_on_glgrid(af_source = sources, af_matrices = parammatrices)

li_dimB = B.dim
li_dimE = E.dim

lpr_sourceE = numpy.zeros(li_dimE ,dtype=numpy.double)
lpr_E = numpy.zeros(li_dimE ,dtype=numpy.double)
lpr_dotE = numpy.zeros(li_dimE ,dtype=numpy.double)
lpr_tmpE = numpy.zeros(li_dimE ,dtype=numpy.double)

lpr_sourceB = numpy.zeros(li_dimB ,dtype=numpy.double)
lpr_B = numpy.zeros(li_dimB ,dtype=numpy.double)
lpr_dotB = numpy.zeros(li_dimB ,dtype=numpy.double)
lpr_tmpB = numpy.zeros(li_dimB ,dtype=numpy.double)

lpi_matrices = (MassB_id,MassE_id,rotational_id,rmatrix_id)
lpi_sources = (S_B.id ,S_E.id )

if ( load_matrices ):
    MassB = mmread( '../Runs/MassB.mtx' )
    MassE = mmread( '../Runs/MassE.mtx' )
    Rotational = mmread( '../Runs/Rotational.mtx' )
    Rmatrix = mmread( '../Runs/Rmatrix.mtx' )

    lpi_matrices = ( )

if ( load_vectors ) :
    lpr_sourceB = genfromtxt ( "../Runs/sourceB.txt" )
```

D.4. isogo class

```
lpr_sourceE = genfromtxt ( "../Runs/sourceE.txt" )

lpi_sources = ()

if ( not load_data ) :

    iso.fem.matrixtoassembly ( lpi_matrices )
    iso.fem.sourcetoassembly ( lpi_sources )

iso.fem.assembly()

if ( not load_matrices ) :
    MassB      = iso.csr_matrix ( MassB_id      )
    MassE      = iso.csr_matrix ( MassE_id      )
    Rotational = iso.csr_matrix ( rotational_id )
    Rmatrix    = iso.csr_matrix ( rmatrix_id    )

if ( not load_vectors ) :
    #computing the projection of the fields over the discrete spaces
    lpr_sourceB = iso.getsource ( S_B )
    lpr_sourceE = iso.getsource ( S_E )

if ( export_matrices ) :
    #———— exporting matrices —————
    mmwrite('../Runs/MassB.mtx', MassB)
    mmwrite('../Runs/MassE.mtx', MassE)
    mmwrite('../Runs/Rotational.mtx', Rotational)
    mmwrite('../Runs/Rmatrix.mtx', Rmatrix)
    #————

if ( export_vectors ) :
    #———— exporting vectors —————
    savetxt("../Runs/sourceB.txt", lpr_sourceB )
    savetxt("../Runs/sourceE.txt", lpr_sourceE )
    #————

#———— factorizing matrices —————
MassE_csc = MassE.tocsc()
op_MassE = splu( MassE_csc )

MassB_csc = MassB.tocsc()
op_MassB = splu( MassB_csc )
#————

lpr_B = op_MassB.solve ( lpr_sourceB )
lpr_E = op_MassE.solve ( lpr_sourceE )

iso.setfield ( B , lpr_B )
iso.setfield ( E , lpr_E )

#diagnostics
if ( not noviz ) :
    iso.silodiag ( exact )

iso.l2errordiag ( exact )

lr_hmax = iso.fem.get_hmax ( )
lr_normerrorB = iso.fem.getnormerror ( B.id, 1 )
lr_normerrorEx = iso.fem.getnormerror ( E.id, 1 )
lr_normerrorEy = iso.fem.getnormerror ( E.id, 2 )

lpr_output = []
lpr_output.append([lr_hmax,lr_normerrorB,lr_normerrorEx,lr_normerrorEy])
ls_file = "error_proj.txt"
savetxt( ls_file , lpr_output )

for i in range(1,niter+1):
```


D.4. isogeo class

```
t = t + dt

if ( timescheme == 2 ) :

    lpr_dotE = Rmatrix.dot ( lpr_B )

    lpr_E = lpr_E + dt * lpr_dotE

    lpr_tmpB = Rotational.dot ( lpr_E )

    lpr_dotB = op_MassB.solve ( lpr_tmpB )

    lpr_B = lpr_B + dt * lpr_dotB

if ( timescheme == 4 ) :

    #-----
    #      Magnetic field evaluation
    #-----
    lpr_dotE      = numpy.zeros(li_dimE ,dtype=numpy.double)
    lpr_tmp1_E    = numpy.zeros(li_dimE ,dtype=numpy.double)
    lpr_tmp2_E    = numpy.zeros(li_dimE ,dtype=numpy.double)

    lpr_dotB      = numpy.zeros(li_dimB ,dtype=numpy.double)
    lpr_tmp1_B    = numpy.zeros(li_dimB ,dtype=numpy.double)
    lpr_tmp2_B    = numpy.zeros(li_dimB ,dtype=numpy.double)
    lpr_tmp3_B    = numpy.zeros(li_dimB ,dtype=numpy.double)

    lpr_tmp1_E = Rmatrix.dot ( lpr_B )

    lpr_tmp1_B = Rotational.dot ( lpr_tmp1_E )

    lpr_tmp2_B = op_MassB.solve ( lpr_tmp1_B )

    lpr_tmp2_E = Rmatrix.dot ( lpr_tmp2_B )

    lpr_dot_E = lpr_tmp1_E + ( 1.0 / 24.0 ) * ( dt **2 ) * lpr_tmp2_E

    lpr_E = lpr_E + dt * lpr_dot_E

    #-----
    #      Magnetic field evaluation
    #-----
    lpr_dotE      = numpy.zeros(li_dimE ,dtype=numpy.double)
    lpr_tmp1_E    = numpy.zeros(li_dimE ,dtype=numpy.double)
    lpr_tmp2_E    = numpy.zeros(li_dimE ,dtype=numpy.double)

    lpr_dotB      = numpy.zeros(li_dimB ,dtype=numpy.double)
    lpr_tmp1_B    = numpy.zeros(li_dimB ,dtype=numpy.double)
    lpr_tmp2_B    = numpy.zeros(li_dimB ,dtype=numpy.double)
    lpr_tmp3_B    = numpy.zeros(li_dimB ,dtype=numpy.double)

    lpr_tmp1_B = Rotational.dot ( lpr_E )

    lpr_tmp2_B = op_MassB.solve ( lpr_tmp1_B )

    lpr_tmp1_E = Rmatrix.dot ( lpr_tmp2_B )

    lpr_tmp3_B = Rotational.dot ( lpr_tmp1_E )

    lpr_tmp1_B = op_MassB.solve ( lpr_tmp3_B )

    lpr_dot_B = lpr_tmp2_B + ( 1.0 / 24.0 ) * ( dt **2 ) * lpr_tmp1_B

    lpr_B = lpr_B + dt * lpr_dot_B

if ( mod ( i , nfreq ) == 0 ):
    print("iteration =" +str(i))
    numdiag = i / nfreq
```

D.5. Using GB-splines

```
iso.setfield ( B , lpr_B )
iso.setfield ( E , lpr_E )

#diagnostics
if ( not noviz ) :
    iso.silodiag ( exact , numdiag )

    iso.l2errordiag ( exact )

lr_normerrorB = iso.fem.getnormerror ( B.id, 1 )
lr_normerrorEx = iso.fem.getnormerror ( E.id, 1 )
lr_normerrorEy = iso.fem.getnormerror ( E.id, 2 )

lpr_output = []
lpr_output.append([lr_hmax,lr_normerrorB,lr_normerrorEx,lr_normerrorEy])
ls_file = "error"+"_"+str(numdiag)+".txt"
savetxt( ls_file , lpr_output )

iso.free ( )
```

D.5 Using GB-splines

TODO

For the moment only few types of GB-splines are implemented. *PyIGA* handles *exp-GB* of degree 4, 5 and variable degree GB-splines *vd-GB* of degree 3.

In the following, we give an example, from `pdelaplace_squareGB.py`, of how we can use GB-splines:

```
iso.fem.set_gbsplines_type(F.id, 2)
lpr_alpha = (5.0,5.0)
iso.fem.set_gbsplines_alpha ( lpr_alpha, F.id, 1 )
```

D.6 PyIGA's available operators

In the sequel we give in detail, the available operators implemented in *PyIGA*.

Mass matrix

The user can create this operator using the *isogeos*' property `iso.MASS`.

The elements of this matrix are,

$$\int_{\Omega} c(x,y)\varphi_b(x,y)\varphi_{b'}(x,y)d\Omega \quad (\text{D.6.1})$$

In this case, the user must give the parameter function c , as in :

```
def parammatrices(work, values):
    values[0,0] = 1.0
```

D.6. PyIGA's available operators

Stiffness matrix

The user can create this operator using the *isogeos*' property **iso.STIFFNESS**.

The elements of this matrix are,

$$\int_{\Omega} \begin{pmatrix} a_{11}(x, y) & a_{12}(x, y) \\ a_{21}(x, y) & a_{22}(x, y) \end{pmatrix} \nabla \varphi_b(x, y) \cdot \nabla \varphi_{b'}(x, y) d\Omega \quad (\text{D.6.2})$$

In this case, the user must give the parameter function matrix A , as in :

```
def parammatrices(work, values):
    values[1, 0] = 1.0
    values[1, 1] = 0.0
    values[1, 2] = 0.0
    values[1, 3] = 1.0
```

the given order must be $a_{11}, a_{12}, a_{21}, a_{22}$.

Advection matrix

The user can create this operator using the *isogeos*' property **iso.ADVECTION**.

$$\int_{\Omega} \mathbf{v} \cdot \nabla \omega_b \quad (\text{D.6.3})$$

where \mathbf{v} is either a given vector, or a numerical field obtained after discretization.

Rotational matrix

The user can create this operator using the *isogeos*' property **iso.ROTATIONAL**.

The elements of this matrix are,

$$\int_{\Omega} \varphi_b \text{rot} \Psi_{b'} d\Omega \quad (\text{D.6.4})$$

Directional derivative matrix

The user can create this operator using the *isogeos*' property **iso.R_MATRIX**.

Derivative with respect to the x axis matrix

The user can create this operator using the *isogeos*' property **iso.DX_MATRIX**.

The elements of this matrix are,

$$\int_{\Omega} c(x, y) \partial_x \varphi_b(x, y) \varphi_{b'}(x, y) d\Omega \quad (\text{D.6.5})$$

Derivative with respect to the y axis matrix

The user can create this operator using the *isogeos*' property **iso.DY_MATRIX**.

The elements of this matrix are,

$$\int_{\Omega} c(x, y) \partial_y \varphi_b(x, y) \varphi_{b'}(x, y) d\Omega \quad (\text{D.6.6})$$

D.7. PyIGA's input data

Poisson's Bracket operator

The user can create this operator using the *isogeo*'s property **iso.POISSON_BRACKET**.

The elements of this matrix are,

$$\int_{\Omega} c(x, y) [\varphi_b, \varphi_{b'}] \varphi_{b'} d\Omega \quad (\text{D.6.7})$$

Weak Rotational matrix

The user can create this operator using the *isogeo*'s property **iso.ROTATIONAL_WEAK**.

The elements of this matrix are,

$$\int_{\Omega} \mathbf{rot} \varphi_b \cdot \Psi_{b'} d\Omega \quad (\text{D.6.8})$$

Mass matrix, in axisymmetric coordinates

The user can create this operator using the *isogeo*'s property **iso.MASS_AXI**.

The elements of this matrix are,

$$\int_{\Omega} c(r, z) \varphi_b(r, z) \varphi_{b'}(r, z) r d\Omega \quad (\text{D.6.9})$$

Rotational matrix, in axisymmetric coordinates

The user can create this operator using the *isogeo*'s property **iso.ROTATIONAL_AXI**.

Weak Rotational matrix, in axisymmetric coordinates

The user can create this operator using the *isogeo*'s property **iso.ROTATIONAL_WEAK_AXI**.

Derivative with respect to the x axis matrix, in axisymmetric coordinates

The user can create this operator using the *isogeo*'s property **iso.DX_AXI_MATRIX**.

Derivative with respect to the y axis matrix, in axisymmetric coordinates

The user can create this operator using the *isogeo*'s property **iso.DY_AXI_MATRIX**.

D.7 PyIGA's input data

D.7.1 Importing domain data

Before executing your test, you must specify the domain you are dealing with. Several examples are given, to create a domain using *NURBS*. To import your domain just add those few lines in the beginning of your script :

D.7. PyIGA's input data

```
import sys

sys.argv=["set_data.py", "Square"]
execfile("/usr/local/pyiga/set_data.py")
```

this will tell *PyIGA* to read data directly using the environment variable *PYIGADATAPATH*. We highly recommend the user to use the option *"-l"*, which allows to read data from *./data*,

```
import sys

sys.argv=["set_data.py", "Square", "-l"]
execfile("/usr/local/pyiga/set_data.py")
```

The user can also import data directly from *Archive_Domains*, this needs to read the environment variable *PYIGAPATH*.

```
import sys

sys.argv=["set_data.py", "Square", "-l"]

PYIGAPATH = os.getenv("PYIGAPATH")
setdatafile=os.path.join(PYIGAPATH, "set_data.py")
execfile(setdatafile)
```

Several domains are given as examples in the archive directory.

D.7.2 Refinement data

Before running your *PyIGA* script, you have to specify some parameters for refinement. This can be done using some specific functions *setters* :

```
iso.fem.set_href ( href )           # h-refinement
iso.fem.set_pref ( pref )           # p-refinement
iso.fem.set_mref ( mref )           # multiplicity
iso.fem.set_ordergl ( ordergl )     # number of Quadrature Points
iso.fem.set_htype ( li_type )       # choose the type of inserted knots
```

h-refinement

PyIGA will insert N_i knots in whole the knot vector, for each vector T_i .

p-refinement

PyIGA will elevate the degree, with the value p_i , for each dimension

D.7. PyIGA's input data

multiplicity

Coupling with a h-refinement process, *PyIGA* will insert N_i knots in whole the knot vector, for each vector T_i with a multiplicity m_i (the same multiplicity for all inserted knots)

type-h-refinement

The user can specify the type of inserted knots. Default treatment, is an equally-spaced knots. (see D.1)

type h insert	corresponding treatment
Default	equally-spaced knots
1	using Gauss-Legendre quadratures points
2	using Gauss-Lobatto quadratures points
3	using a decentered version of Gauss-Legendre quadratures points
4	using Gauss-Legendre quadratures points, by inversion the order
5	using Hammersley sequence

Figure D.1: h-refinement types

D.7.3 Boundary conditions

In (D.2), we give the different values for this parameter. Not all cases are covered for the moment.

Boundary Condition	corresponding treatment on ξ	corresponding treatment on η
0	free	free
1	periodic	free
-1	periodic maximum regularity	free
2	free	periodic
-2	free	periodic maximum regularity
3	periodic	periodic
10	dirichlet	dirichlet
11	dirichlet	free
12	free	dirichlet
21	dirichlet	\mathcal{C}^0 periodicity
22	\mathcal{C}^0 periodicity	dirichlet
111	free	North = free, South = dirichlet

Figure D.2: Boundary condition values

D.8. Creating domains using the GUI

D.7.4 Silver Muller condition

D.7.5 Details and output file

D.8 Creating domains using the GUI

D.8.1 The XML Format

We have adopted the following norm for domain descriptions.

Code Listing D.6: XML description for the circle

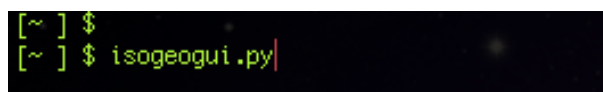
```
<xml>
<patch>
<param_domain>
<n>3,3</n>
<p>2,2</p>
</param_domain>
<knots>
0.0, 0.0, 0.0, 1.0, 1.0, 1.0
</knots>
<knots>
0.0, 0.0, 0.0, 1.0, 1.0, 1.0
</knots>
<points>
-0.35355339059327379, -0.35355339059327379, 1.0;
-0.70710678118654746, 0.0, 0.70710678118654752440;
-0.35355339059327379, 0.35355339059327379, 1.0;
0.0, -0.70710678118654746, 0.70710678118654752440;
0.0, 0.0, 1.0;
0.0, 0.70710678118654746, 0.70710678118654752440;
0.35355339059327379, -0.35355339059327379, 1.0;
0.70710678118654746, 0.0, 0.70710678118654752440;
0.35355339059327379, 0.35355339059327379, 1.0
</points>
</patch>
</xml>
```

Remark D.8.1 *As one can see for the control points, the last column contains the associated weights.*

Remark D.8.2 *This description allows us to use multiple-patches.*

D.8.2 Using the GUI

By executing the following command in your shell (see figure D.3), you may access to the *PyIGA-GUI*.



```
[~ ] $
[~ ] $ isogeogui.py
```

Figure D.3: How to run the GUI

D.8.3 Examples

Example 1

In this example we show how to construct a domain from an initial square. For this, we can clic on the *open* button, a new window appears, that allows the user to browse the *Archive domains* directory in *pyiga*'s home (figure D.4):

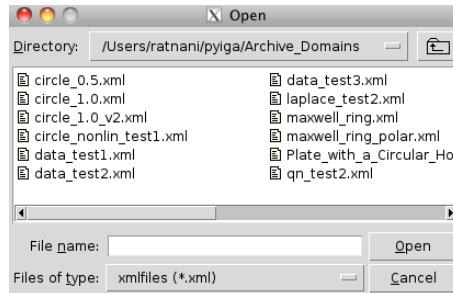


Figure D.4: Initializing the patch, by choosing a initial configuration

In the figure D.5, we show the different steps of such construction:

1. we choose a square as an initial configuration,
2. we clic on *update* button to draw the lines,
3. we move one control point,
4. we elevate the degree of the curve by 1; the corresponding curve is quadratic,
5. we move those new control points, corresponding to the middle of the exterior edges,
6. we use an h-refinement, by inserting 8 knots in each direction.

After each step, the user must click on the *update* button.

Example 2

To create a circle, the user can choose it in (figure D.4). This example shows the meshing of a circle (figure D.6).

Example 3

In this example, we show how one can construct a compact convex domain (figure D.7). In figure D.8, we give examples of domains where boundaries are defined implicitly. In this case, we only approach the boundary (we must use the isoparametric version of IGA).

D.9. Creating domains defined by an implicit function

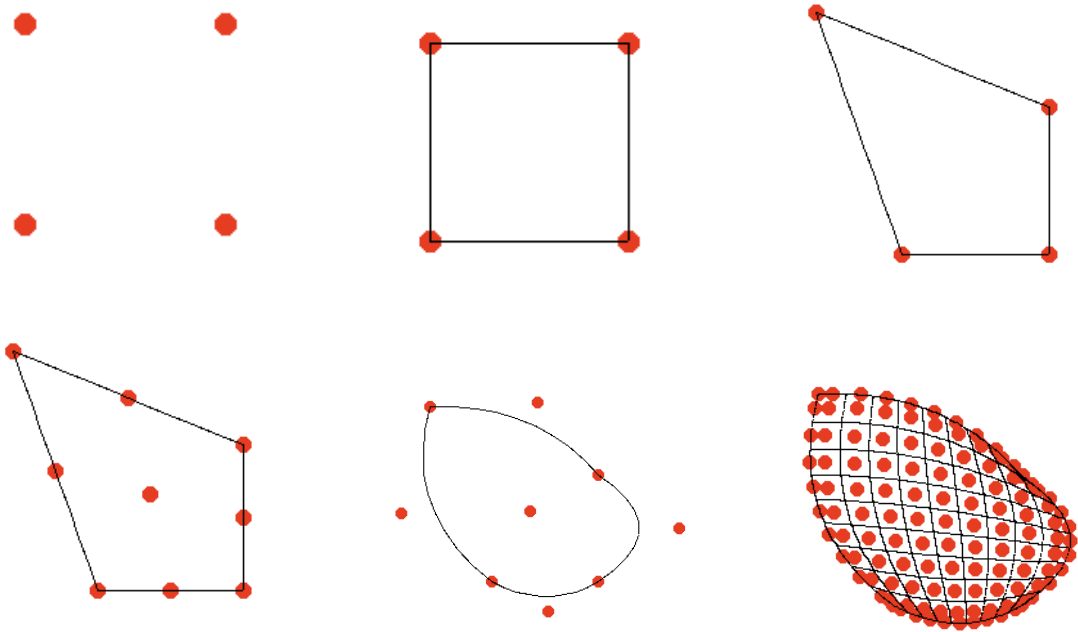


Figure D.5: Creating and Meshing a domain

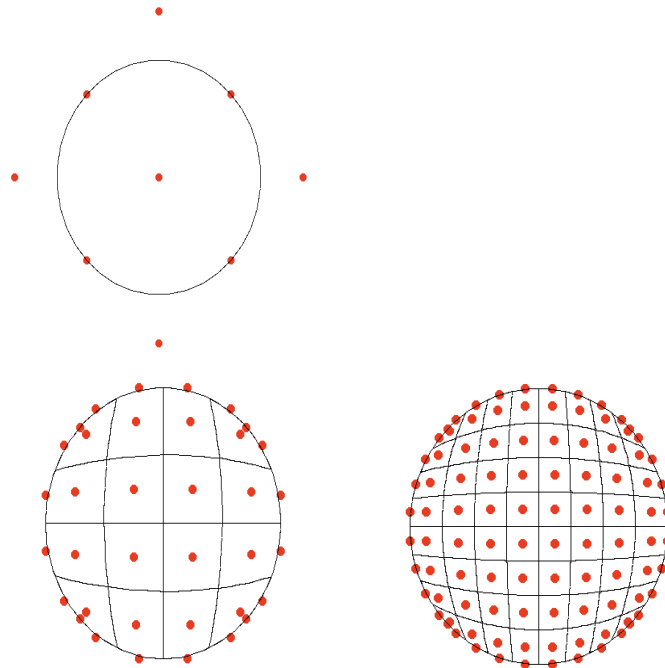


Figure D.6: The circle after h-refinement

D.9 Creating domains defined by an implicit function

In this section, we show how to construct a domain defined by an implicit function, and generate an adequate meshing. This is done thanks to the routines `genermesh_mhdeq`.

D.9. Creating domains defined by an implicit function

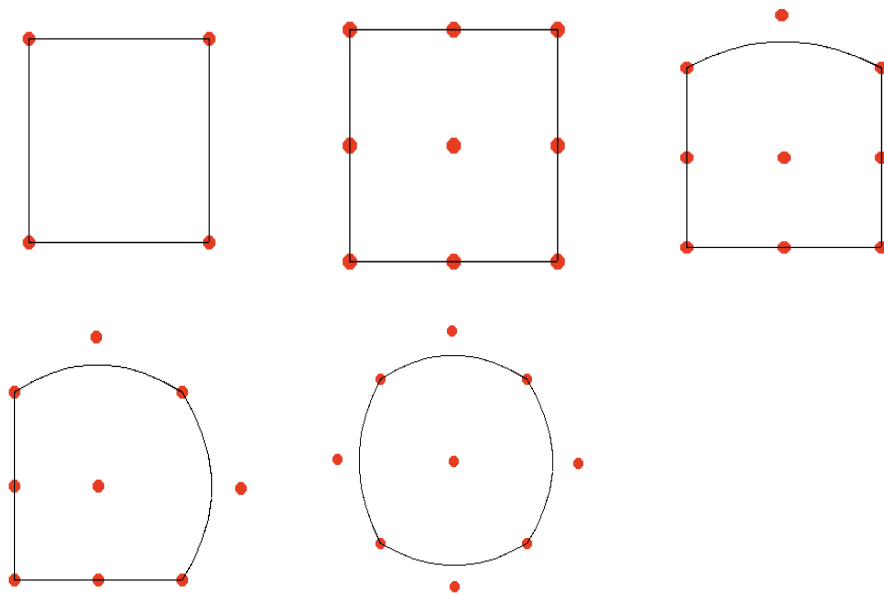


Figure D.7: Parametrization of the circle; starting from a square, we move some of the control points

Here is an example of such treatment (taken from `pdesoloviev.py`).

```
#using genermesh_mhdeq_v2
#param using genermesh_mhdeq_v2
from genermesh_mhdeq_v2 import *
n = 32
levelmax = 0.975
typelevel = 0
genermesh_mhdeq (n, testcase.psi, testcase.dxppsi, testcase.dyppsi, ↵
    testcase.phi, testcase.dphi, levelmax, typelevel)
```

The user can also another version of `genermesh_mhdeq`, but it is less efficient than the previous one.

```
#using genermesh_mhdeq_v1
#param using genermesh_mhdeq_v1
from genermesh_mhdeq_v1 import *
n = 32
prof = 4
r = 0.7
genermesh_mhdeq ( n , prof , r , testcase.psi , testcase.dxppsi , ↵
    testcase.dyppsi )
```

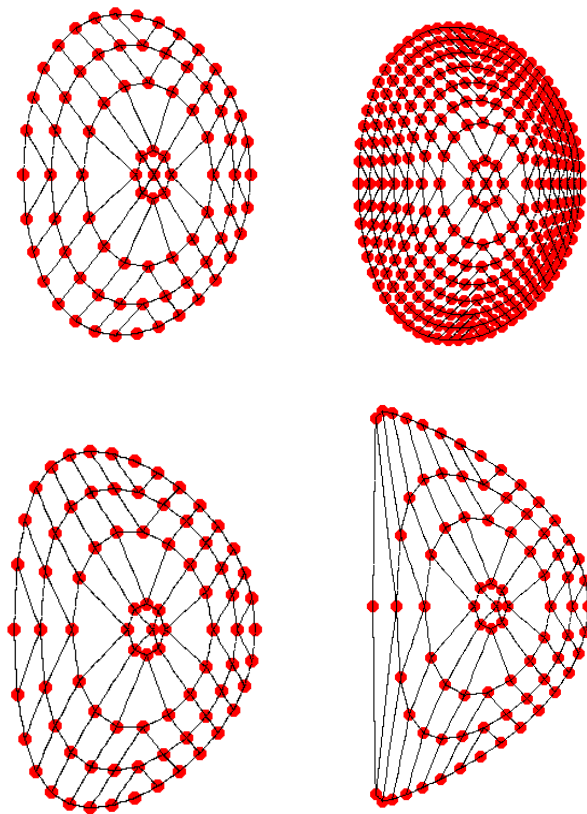


Figure D.8: Soloviev domains: (first line) for test1 using splines of degree 1 and (left) 8×8 meshes, (right) 16×16 meshes, (second line) (left) test4, (right) test6 using splines of degree 1 and (left) 8×8 meshes

D.10 Visualization

D.10.1 Using Silo

Visualizations can be done using *VisIt*. The user can specify the desired detail for this diagnostic.

For visualization, there are only two parameters to control diagnostics `gi_npoint_SILO` and `gi_detail_SILO`. For a better image quality, the user may increase the value of `gi_npoint_SILO`; this is the number of intervals in each mesh $[t_i, t_{i+1}]$ for vector knots. When the h-refinement parameter h is small, we can take it equal to 1.

To control those parameters, the user must call,

```
iso.fem.set_npoint_silo ( 1 )      # the resolution of diagnostics
iso.fem.set_detail_silo ( 4 )     # the detail level of diagnostics
```

Depending on the value of `gi_detail_SILO`, *PyIGA* may give certain details (please see D.9).

The user may also want a diagnostic in the patch, this can be done by adding 10 to the desired value of `gi_detail_SILO` . Diagnostics will be given in the physical domain and

D.11. PyIGA and Pastix

gi_detail_SILO	corresponding treatment
1	numerical solution of the pde, namely u_h
2	u_h and the jacobian
3	u_h , the jacobian and control points
4	u_h , the jacobian, control points, the field u and $u - u_h$

Figure D.9: Visualization details

the patch.

D.10.2 Using VTK

TODO

D.11 PyIGA and Pastix

To use *Pastix*, we need to load the following module,

```
module load openmpi-x86_64
```

we have created a *pastrix* class, that allows us to communicate directly with *isogeo*.

```
from pastrix import *

#--- Constructing the Pastrix object
Stiff_past = pastrix ( pastix, iso.fem, stiffness_id )

#--- Solving the linear system
lpr_tmp = lpr_source
Stiff_past.solve ( lpr_tmp )
```

D.12 Installing PyIGA

TODO The user must define the environment variable **PYIGADATAPATH**, which defines the directory where *PyIGA* will read input data.

The user must define the environment variable **PYIGAWORKPATH**, which defines the working directory for *PyIGA*. He must respect the following tree:

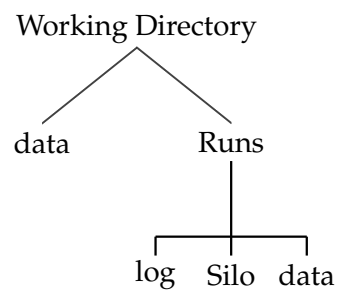


Figure D.10: Working directory tree

Bibliography

- [1] J. Abiteboul, G. Latu, V. Grandgirard, A. Ratnani, E. Sonnendrücker, and A. Strugarek. Solving the vlasov equation in complex geometries. Proceedings of CEM-RACS 2010, submitted.
- [2] M. Aigner, C. Heinrich, B. Jüttler, E. Pilgerstorfer, B. Simeon, and A. V. Vuong. Swept volume parameterization for isogeometric analysis. In *Proceedings of the 13th IMA International Conference on Mathematics of Surfaces XIII*, pages 19–44, Berlin, Heidelberg, 2009. Springer-Verlag.
- [3] T.S. Hahm A.J. Brizard. Foundations of nonlinear gyrokinetic theory. *Reviews of Modern Physics*, 79:421–468, 2007. <http://www.damtp.cam.ac.uk/user/tong/dynamics.html>.
- [4] P. Angelino, X. Garbet, and *al.* Role of Plasma Elongation on Turbulent Transport in Magnetically Confined Plasmas. *Physical Review Letters*, 102(19), 2009.
- [5] Douglas N. Arnold, Daniele Boffi, and Richard S. Falk. Quadrilateral h(div) finite elements. *SIAM J. Numer. Anal.*, 42:2429–2451.
- [6] Franck Assous, Patrick Ciarlet, and Simon Labrunie. Theoretical tools to solve the axisymmetric maxwell equations. *Mathematical Methods in the Applied Sciences*, 25:49–78, 2002.
- [7] Franck Assous, Patrick Ciarlet, and Simon Labrunie. Solution of axisymmetric maxwell equations. *Mathematical Methods in the Applied Sciences*, 26:861–896, 2003.
- [8] A. Back, A. Crestetto, A. Ratnani, and E. Sonnendrücker. Isopic: coupling an axisymmetric pic solver with the isogeometric approach. In *In Proceedings of ESAIM*, 2010. submitted.
- [9] P.M. Bellan. *Fundamentals of Plasma Physics*. cambridge university press, first edition, 2006.
- [10] 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.
- [11] C. K. Birdsall and A. B. Langdon. *Plasma Physics Via Computer Simulation*. Institute of Physics Publishing, Bristol and Philadelphia, 2002.
- [12] C.K. Birdsall and A. Langdon. *Plasma physics via computer simulation*. McGraw-Hill, New York, NY, USA, 1985.

Bibliography

- [13] A. Bossavit. Mixed finite elements and the complex of Whitney forms. *The Mathematics of Finite Elements and Applications VI*, J. Whiteman (ed.), pages 137–144, 1988.
- [14] A. Bossavit. Computational electromagnetism and geometry : building a finite-dimensional “Maxwell’s house” (part 5). *J. Japan Soc. Applied Electromagnetism and Mechanics*, 8:203–209, 2000.
- [15] A. Buffa, C. de Falco, and G. Sangalli. Isogeometric analysis: Stable elements for the 2d stokes equation. *International Journal for Numerical Methods in Fluids*, 65(11-12):1407–1422, 2011.
- [16] A. Buffa, J. Rivas, G. Sangalli, and R. Vázquez. Isogeometric discrete differential forms in three dimensions. *SIAM J. Numerical Analysis*, 49(2):818–844, 2011.
- [17] A. Buffa, J. Rivas, G. Sangalli, and R. Vázquez. Isogeometric analysis in electromagnetics: theory and testing. Technical report, IMATI-CNR.
- [18] A. Buffa, G. Sangalli, and R. Vázquez. Isogeometric analysis in electromagnetics: B-splines approximation. *Comput. Methods Appl. Mech. Engrg*, 199:1143–1152, 2009.
- [19] B. Swartz C. de Boor. Collocation at gaussian points. *SIAM J. Numer. Anal.*, 19, 1973.
- [20] P. G. Ciarlet and P. A. Raviart. Interpolation theory over curved elements, with applications to finite element methods. *CMAME*, 1:217–249, 1972.
- [21] E. Cohen, T. Martin, R.M. Kirby, T. Lyche, and R.F. Riesenfeld. Analysis-aware modeling: Understanding quality considerations in modeling for isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):334 – 356, 2010. Computational Geometry and Analysis.
- [22] Paolo Costantini, Carla Manni, Francesca Pelosi, and M. Lucia Sampoli. Quasi-interpolation in isogeometric analysis based on generalized b-splines. *Comput. Aided Geom. Des.*, 27:656–668, November 2010.
- [23] J.A Cottrell, T. Hughes, and Y. Bazilevs. *Isogeometric Analysis, toward Integration of CAD and FEA*. John Wiley & Sons, Ltd, first edition, 2009.
- [24] N. Crouseilles, G. Latu, and E. Sonnendrücker. Hermite spline interpolation on patches for parallelly solving the Vlasov-Poisson equation. *Int. J. of Applied Math. and Computer Science*, 17(3):335–349, 2007.
- [25] N. Crouseilles, G. Latu, and E. Sonnendrücker. A parallel Vlasov solver based on local cubic spline interpolation on patches. *Journal of Computational Physics*, 228(5):1429–1446, 2009.
- [26] H.B. Curry and I.J. Schoenberg. On polya frequency functions iv: The fundamental spline functions and their limits. *J. d’Analyse Math*, 17:71–107, 1966.
- [27] Olivier Czarny and Guido Huysmans. Bézier surfaces and finite elements for mhd simulations. *J. Comput. Phys.*, 227:7423–7445, August 2008.
- [28] Polyanin A. D. and Zaitsev V. F. *Handbook of Nonlinear Partial Differential Equations*. Chapman, Hall CRC, 2004. BocaRaton.

Bibliography

- [29] D. Biskamp. *Nonlinear Magnetohydrodynamics*. Cambridge University Press, 1997.
- [30] C. de Boor, K. Höllig, and S. Riemenschneider. *Box splines*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.
- [31] C. DeBoor. *A practical guide to splines*. Springer-Verlag, New York, applied mathematical sciences 27 edition, 2001.
- [32] E. Deriaz, B. Depres, G. Faccanoni, K.P. Gostaf, L.M. Imbert-Gerard, G. Sadaka, and R. Sart. Magnetic equations with freefem++: The grad-shafranov equation and the current hole. In *In Proceedings of ESAIM*, 2010. submitted.
- [33] R.A. DeVore and G.G. Lorentz. *Constructive Approximation*. Springer-Verlag, Berlin, Heidelberg, 1993.
- [34] D.H.E. Dubin, J. A. Krommes, C. Oberman, and W. W. Lee. Nonlinear gyrokinetic equations. *Phys. Fluids*, 26, 1983.
- [35] Michael R. Dörfel, Bert Jüttler, and Bernd Simeon. Adaptive isogeometric analysis by local h-refinement with t-splines. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):264 – 275, 2010. Computational Geometry and Analysis.
- [36] B. DÈpres and R. Sart. Reduced resistive mhd with general density i: Model and stability results. 2011. Technical Report RR11003, LJLL, UPMC.
- [37] Lynch Robert E., Rice John R., and Thomas Donald H. Direct solution of partial difference equations by tensor product methods. *Numerische Mathematik*, 6:185–199, 1964. 10.1007/BF01386067.
- [38] J. W. Eastwood, W. Arter, N. J. Brealey, and R. W. Hockney. Body-fitted electromagnetic pic software for use on parallel computers. *Computer Physics Communications*, 87(1-2):155 – 178, 1995. Particle Simulation Methods.
- [39] John A. Evans, Yuri Bazilevs, Ivo Babuska, and Thomas J.R. Hughes. n-widths, sup-infs, and optimality ratios for the k-version of the isogeometric finite element method. *Computer Methods in Applied Mechanics and Engineering*, 198(21-26):1726 – 1741, 2009. Advances in Simulation-Based Engineering Sciences - Honoring J. Tinsley Oden.
- [40] Hassan Fahs and Stéphane Lanteri. A high-order non-conforming discontinuous galerkin method for time-domain electromagnetics. *J. Comput. Appl. Math.*, 234:1088–1096, June 2010.
- [41] G. Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann Pub. Inc., San Francisco, CA, USA, 2002.
- [42] F. Filbet and E. Sonnendrücker. Modeling and numerical simulation of space charge dominated beams in the paraxial approximation. *Mathematical Models and Methods in Applied Sciences*, 16(5):763, 2006.
- [43] F. Filbet, E. Sonnendrücker, and P. Bertrand. Conservative Numerical Schemes for the Vlasov Equation. *Journal of Computational Physics*, 172(1):166 – 187, 2001.

Bibliography

- [44] Paul F. Fischer. Anisotropic diffusion in a toroidal geometry. *Journal of Physics: Conference Series*, 16:446–455, 2005.
- [45] X. Garbet, Y. Idomura, L. Villard, and T.-H. Watanabe. Gyrokinetic simulations of turbulent transport. *Nuclear Fusion*, 50(4):043002, 2010.
- [46] M. Geimer and O. Abert. Interactive ray tracing of trimmed bicubic bézier surfaces without triangulation. In *Proceedings of WSCG*, pages 71–78, 2005.
- [47] A.H. Glasser, I.A. Kitaeva, V.D. Liseikin, V.S. Lukin, and A.N. Simakov. Harmonic grid generation for the tokamak edge region. In *Proceedings of EPS, Spain*, 2005.
- [48] Mark S. Gockenbach. *Understanding and Implementing the Finite Element Method*. SIAM, first edition, 2006.
- [49] Herbert Goldstein. *Classical mechanics*. Addison-Wesley Publishing Co., Reading, Mass., second edition, 1980. Addison-Wesley Series in Physics.
- [50] H. Grad, A. Kadish, and D. C. Stevens. A free boundary tokamak equilibrium. *Communications on Pure and Applied Mathematics*, 27(1):39–57, 1974.
- [51] V. Grandgirard, M. Brunetti, P. Bertrand, N. Besse, X. Garbet, P. Ghendrih, G. Manfredi, Y. Sarazin, O. Sauter, E. Sonnendrücker, J. Vaclavik, and L. Villard. A drift-kinetic Semi-Lagrangian 4D code for ion turbulence simulation. *Journal of Computational Physics*, 217(2):395–423, 2006.
- [52] V. Grandgirard, M. Brunetti, P. Bertrand, N. Besse, X. Garbet, P. Ghendrih, G. Manfredi, Y. Sarazin, O. Sauter, E. Sonnendrücker, J. Vaclavik, and L. Villard. A drift-kinetic Semi-Lagrangian 4D code for ion turbulence simulation. *Journal of Computational Physics*, 217(2):395–423, 2006.
- [53] V. Grandgirard, Y. Sarazin, P. Angelino, A. Bottino, N. Crouseilles, G. Darnet, G. Dif-Pradalier, X. Garbet, P. Ghendrih, S. Jolliet, et al. Global full-f gyrokinetic simulations of plasma turbulence. *Plasma Phys. and Control. Fusion*, 49:B173, 2007.
- [54] F.W. Perkins G.W. Hammett. Fluid models for landau damping with application to the ion-temperature-gradient instability. *Phys. Rev. Lett.*, 64:3019–3022, 1990.
- [55] T.S. Hahm. Nonlinear gyrokinetic equations for tokamak microturbulence. *Phys. Fluids*, 31, 1988.
- [56] Dimits Beer Hammett, A. M. Dimits, M. A. Beer, G. W. Hammett, S. E. Parker, A. J. Redd, and J. Weiland. Comparisons and physics basis of tokamak transport models and turbulence simulations. *Phys. Plasmas*, 7:969–983, 2000.
- [57] R. Hatzky, T.M. Tran, A. Koenis, R. Kleiber, and S.J. Allfrey. Energy conservation in a nonlinear gyrokinetic particle-in-cell code for ion-temperature-gradient-driven modes in θ -pinch geometry. *Phys. Plasmas*, 9, 2002.
- [58] K. Höllig. *Finite Element Methods with B-Splines*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.
- [59] K. Höllig. *Finite Element Methods with B-Splines*, volume 26. SIAM, 2003. Frontiers in Applied Mathematics.

Bibliography

- [60] K Höllig and DJ Benson. Finite element methods with b-splines. *Applied Mechanics Reviews*, 57, 2004.
- [61] K. Höllig, Reif Ulrich, and Wipper Joachim. Multigrid methods with b-splines. *Numerische Mathematik*, 91:237–255, 2002.
- [62] Klaus Höllig, Jörg Hörner, and Martina Pfeil. Parallel finite element methods with weighted linear b-splines. pages 667–676, 2008.
- [63] Klaus Höllig, Ulrich Reif, and Joachim Wipper. Weighted extended b-spline approximation of dirichlet problems. *SIAM J. Numer. Anal.*, 39:442–462, February 2001.
- [64] Qi-Xing Huang, Shi-Min Hu, and Ralph R. Martin. Fast degree elevation and knot insertion for b-spline curves. *Computer Aided Geometric Design*, 22(2):183 – 197, 2005.
- [65] T. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications Inc., 2003.
- [66] T.J.R. Hughes, Y. Bazilevs, L. Beirao de Veiga, J.A. Cottrell, and G. Sangalli. Isogeometric analysis: approximation, stability and error estimates for h-refined meshes. *Mathematical Models and Methods in Applied Sciences (M3AS)*, 16:1031–1090, 2006.
- [67] T.J.R. 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(39-41):4135 – 4195, 2005.
- [68] T.J.R. Hughes, A. Reali, and G. Sangalli. Efficient quadrature for nurbs-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):301 – 313, 2010. Computational Geometry and Analysis.
- [69] K. Höllig, J. Hörner, and M. Pfeil. *Parallel Finite Element Methods with Weighted Linear B-Splines*. Springer Berlin Heidelberg, 2008.
- [70] B.M. Irons. Engineering application of numerical integration in stiffness method. *Journal of the American Institute of Aeronautics and Astronautics*, 4:2035–2037, 1966.
- [71] S. Jardin. *Computational Methods in Plasma Physics*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2010.
- [72] Rong-Qing Jia. Local linear independence of the translates of a box spline. *Constructive Approximation*, 1:175–182, 1985. 10.1007/BF01890029.
- [73] S. Jolliet, A. Bottino, P. Angelino, R. Hatzky, TM Tran, BF McMillan, O. Sauter, K. Appert, Y. Idomura, and L. Villard. A global collisionless PIC code in magnetic coordinates. *Computer Physics Communications*, 177(5):409–425, 2007.
- [74] Höllig Klaus, Reif Ulrich, and Wipper Joachim. Weighted extended b-spline approximation of dirichlet problems. *SIAM J. Numer. Anal.*, 39:442–462, February 2001.
- [75] Boris I. Kvasov. Shape preserving c^2 spline interpolation. In *proceedings of the second asian mathematical conference 1995*, 1998.

Bibliography

- [76] Boris I. Kvasov. Approximation by discrete gb-splines. *Numerical Algorithms*, 27:169–188, 2001. 10.1023/A:1011818621589.
- [77] W. Tiller L. Piegl. *The NURBS Book*. Springer-Verlag, Berlin, Heidelberg, 1995. second ed.
- [78] Ming-Jun Lai and Paul Wenston. Bivariate splines for fluid flows. *Computers and Fluids*, 33(8):1047 – 1073, 2004.
- [79] H. P. Langtangen. *Python Scripting for Computational Science*. Springer-Verlag Berlin Heidelberg, third edition, 2008.
- [80] W.W. Lee. Gyrokinetic approach in particle simulation. *Phys. Fluids*, 26, 1983.
- [81] M. Lenoir. Optimal isoparametric finite elements and error estimates for domains involving curved boundaries. *SIAM J. Numer. Anal.*, 23:562–580, June 1986.
- [82] D. Lischinski and J. Gonczarowski. Improved techniques for ray tracing parametric surfaces. *The Visual Computer*, 6:134–152, 1990.
- [83] W. Ma and J. P. Kruth. Nurbs curve and surface fitting for reverse engineering. *The International Journal of Advanced Manufacturing Technology*, 14:918–927, 1998. 10.1007/BF01179082.
- [84] T. Martin, E. Cohen, and R.M. Kirby. Volumetric parameterization and trivariate b-spline fitting using harmonic functions. *Computer Aided Geometric Design*, 26(6):648 – 664, 2009. Solid and Physical Modeling 2008, ACM Symposium on Solid and Physical Modeling and Applications.
- [85] Arnold Douglas N., Falk Richard S., and Winther Ragnar. Finite element exterior calculus, homological techniques, and applications. *Acta Numer.*, 15:1–155, 2006.
- [86] Goldman R. N. and Lyche T. *Knot Insertion and Deletion Algorithms for B-Spline Curves and Surfaces*. SIAM, Philadelphia, USA, 1993.
- [87] Marian Neamtu. Bivariate simplex b-splines: A new paradigm. In *Proceedings of the 17th Spring conference on Computer graphics*, pages 71–, Washington, DC, USA, 2001. IEEE Computer Society.
- [88] Y. Nishimura, Z. Lin, J.L.V. Lewandowski, and S. Either. A finite element poisson solver for gyrokinetic particle simulations ina global field aligned mesh. *J. Comput. Phys.*, 214:657–671, 2006.
- [89] T. Nishita, T. W. Sederberg, and M. Kakimoto. Ray tracing trimmed rational surface patches. *SIGGRAPH Comput. Graph.*, 24(4):337–345, 1990.
- [90] Monk P. *Finite Element Methods for Maxwell’s Equations*. Calderon Press (Oxford), 2003.
- [91] L.A. Piegl and W. Tiller. *The NURBS book*. Springer Verlag, 1997.
- [92] D.E. Baldwin P.J. Catto, W.M. Tang. Generalized gyrokinetics. *Plasma Phys*, 23:639, 1981.

Bibliography

- [93] Hartmut Prautzsch and Bruce Piper. A fast algorithm to raise the degree of spline curves. *Comput. Aided Geom. Des.*, 8:253–265, October 1991.
- [94] H. Qin. A short introduction to general gyrokinetic theory. *PPPL*, 2005. report 4052.
- [95] Hiptmair R. Finite elements in computational electromagnetism. *Acta Numerica*, 11:237–339, 2002.
- [96] J.D. Meiss R.D. Hazeltine. *Plasma Confinement*. Dover edition, 2003.
- [97] Wolfgang Heidrich Richard, Richard Bartels, and George Labahn. Fitting uncertain data with nurbs. In *Proc. 3rd Int. Conf. on Curves and Surfaces in Geometric Design*, Vanderbilt University Press, 1997.
- [98] L. L. Schumaker. *Spline Functions: Basic Theory*. Wiley (New York), 1981.
- [99] L.L. Schumaker and M-J. Lai. *Spline Functions on Triangulations*, volume Encyclopedia of Mathematics and its Applications 110. Cambridge press, 2007.
- [100] T.W. Sederberg, D.L. Cardon, J. Zheng, and T. Lyche. T-spline simplification and local refinement. *ACM Trans, Graphics*, 23:276–283, 2004.
- [101] P. Sharma and G.W. Hammett. A fast semi-implicit method for anisotropic diffusion. *Journal of Computational Physics*, 2011. Accepted for publication.
- [102] B. Smits. Efficiency issues for ray tracing. *J. Graph. Tools*, 3:1–14, February 1998.
- [103] E. Sonnendrücker. Modèles cinétiques pour la fusion. notes du cours de M2 (2008).
- [104] E. Sonnendrücker, F. Filbet, A. Friedman, E. Oudet, and J.-L. Vay. Vlasov Simulations of beams with a moving grid. *Computer Physics Communications*, 164(1–3):390–395, 2004.
- [105] E. Sonnendrücker, M. Gutnic, M. Haefele, G. Latu, and J.L. Lemaire. Vlasov Simulation of Beams and HALO. In *Proceedings of the Particle Accelerator Conference, 2005*, pages 581–585, 2005.
- [106] 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.
- [107] A. Staniforth and J. Côté. Semi-lagrangian integration schemes for atmospheric models: A review. *Monthly Weather Review*, 119(9):2206–2223, 1991.
- [108] Dokken T. Workshop on: "Non-Standard Numerical Methods for PDE's", Pavia, Italy, jun 29 - jul 02.
- [109] I.C. Taig. Structural analysis by the matrix displacement method. *English Electric Aviation Report*, SO 17, 1961.
- [110] David Tong. Lectures on classical dynamics. <http://www.damtp.cam.ac.uk/user/tong/dynamics.html>.
- [111] D. L. Toth. On ray tracing parametric surfaces. *SIGGRAPH Comput. Graph.*, 19(3):171–179, 1985.

Bibliography

- [112] K. G. van der Zee and C. V. Verhoosel. Isogeometric analysis-based goal-oriented error estimation for free-boundary problems. *Finite Elem. Anal. Des.*, 47:600–609, June 2011.
- [113] Charles F. van Loan. The ubiquitous kronecker product. *J. Comput. Appl. Math.*, 123:85–100, November 2000.
- [114] A.-V. Vuong, C. Giannelli, B. Jüttler, and B. Simeon. A hierarchical approach to local refinement in isogeometric analysis. Submitted.
- [115] S.W. Wang, Z.C. Shih, and R.C. Chang. An efficient and stable ray tracing algorithm for parametric surfaces. *J. Inf. Sci. Eng.*, 18(4):541–561, 2002.
- [116] R. A. Kolesnikov W.W. Lee. On high-order corrections to gyrokinetic vlasov-poisson equations in the long wavelength limit. 2009. PPPL- 4382 report.
- [117] Gu Xianfeng, He Ying, and Qin Hong. Manifold splines. In *Proceedings of the 2005 ACM symposium on Solid and physical modeling, SPM '05*, pages 27–38, New York, NY, USA, 2005. ACM.
- [118] G. Xu, B. Mourrain, R. Duvigneau, and A. Galligo. Optimal analysis-aware parameterization of computational domain in isogeometric analysis. submitted.