Aix*Marseille université
Socialement engagée

NNT/NL :
2020AIXM0001/001ED000

THÈSE DE DOCTORAT

Soutenue à Aix-Marseille Université
le 2 décembre 2022 par

Emily Bourne

Non-Uniform Numerical Schemes for the Modelling of Turbulence in the 5D GYSELA Code

Discipline
Mathématiques

Spécialité
Analyse Appliquée

École doctorale
ED 184 MATHEMATIQUES ET INFORMATIQUE

Laboratoire/Partenaires de recherche
renseigner les partenaires institutionnels
et les partenaires privés
un partenaire par ligne

Composition du jury

Bruno DESPRÉS LJLL Sorbonne University	Rapporteur
Virginie EHRLACHER CERMICS, ENPC	Rapporteure
Philippe HELLUY Université de Strasbourg	Examinateur
Carola KRUSE CERFACS	Examinatrice
Claudia NEGULESCU Université de Toulouse	Examinatrice
Eric SONNENDRÜCKER Max-Planck Institut für Plasmaphysik	Président du jury
Michel MEHRENBERGER I2M	Directeur de thèse
Virginie GRANDGIRARD CEA	Co-Directrice de thèse

Affidavit

I, undersigned, Emily Bourne, hereby declare that the work presented in this manuscript is my own work, carried out under the scientific direction of Virginie Grandgirard and Michel Mehrenberger, in accordance with the principles of honesty, integrity and responsibility inherent to the research mission. The research work and the writing of this manuscript have been carried out in compliance with both the french national charter for Research Integrity and the Aix-Marseille University charter on the fight against plagiarism.

This work has not been submitted previously either in this country or in another country in the same or in a similar version to any other examination body.

Marseille, 19/09/2022

List of publications and participation in conferences

List of publications written as part of the thesis project:

1. Emily Bourne, Yann Munschy, Virginie Grandgirard, et al. “Non-Uniform Splines for Semi-Lagrangian Kinetic Simulations of the Plasma Sheath”. In: *Journal of Computational Physics* (Aug. 2022). under review (https://track.authorhub.elsevier.com/?uuid=b7ad331b-bd6c-4e85-834d-2c117471df74). URL: https://hal-cea.archives-ouvertes.fr/cea-03748016

2. Emily Bourne, Philippe Leleux, Katharina Kormann, et al. “Solver comparison for Poisson-like equations on tokamak geometries”. In: *Journal of Computational Physics* (Sept. 2022). submitted. URL: https://hal-cea.archives-ouvertes.fr/cea-03786723

3. Emily Bourne, Yaman Güçlü, Said Hadjout, et al. “Pyccel: a Python-to-X transpiler for scientific high-performance computing”. In: *Journal of Open Source Software* (Nov. 2022). under review

List of publications co-authored as part of the thesis project:

1. Yann Munschy, Emily Bourne, Guilhem Dif-Pradalier, et al. “Kinetic plasma-wall interaction using immersed boundary conditions”. in preparation. 2022

2. Xavier Garbet, Olivier Panico, R. Varennes, et al. “Zonal instability and wave trapping”. In: *Journal of Physics: Conference Series* 1785 (Feb. 2021), p. 012002. DOI: 10.1088/1742-6596/1785/1/012002

3. R. Varennes, X. Garbet, L. Vermare, et al. “Synergy of Turbulent Momentum Drive and Magnetic Braking”. In: *Phys. Rev. Lett.* 128 (25 June 2022), p. 255002. DOI: 10.1103/PhysRevLett.128.255002. URL: https://link.aps.org/doi/10.1103/PhysRevLett.128.255002

4. R Varennes, X Garbet, L. Vermare, et al. “Impact of magnetic ripple on neoclassical equilibrium in gyrokinetic simulations”. working paper or preprint. Apr. 2022. URL: https://hal.archives-ouvertes.fr/hal-03631218

Participation in conferences and summer schools during the duration of the thesis:

1. Presentation entitled “A semi-Lagrangian scheme with non-equidistant splines to investigate sheath physics” given at the NumKin 2020 workshop
2. Participated in the research school "Scaling Limits from Microscopic to Macroscopic Physics" 2021
3. Participated in the Culham Summer School 2021
4. Participated in the CEMRACS Summer School 2022

Summary

Predicting the performance of a fusion plasma as a function of the plasma power gains, is one of the crucial challenges in fusion plasma physics. With that in mind, turbulence and heat transport must be modelled in a precise theoretical framework which in this case involves the use of "first principle" non-linear simulations. The 5D (3 spatial coordinates, 2 velocity coordinates) gyrokinetic equations for each species (ions and electrons), coupled with the 3D Maxwell equations form a self-consistent description of the problem. Studies of transport in the core of a tokamak plasma have now reached maturity. Several first principle codes exist which are capable of handling this problem. However, despite their many successes, their prediction capabilities remain limited by the energy content, in particular in the case of discharges with optimised confinement time. In order to push past this limitation, the gyrokinetic models must be extended towards the region at the edge of a tokamak, and, where possible, should treat the transport at the edge and at the core in the same way.

The 5D gyrokinetic non-linear GYSELA (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016) code developed by IRFM/CEA is special in that it is global (simulates the entire torus), makes no scale separation approximations ("full F code") and drives turbulence via particle, momentum and heat sources. The additional use of an immersed boundary condition, imitating the extraction of heat by a limiter, makes the GYSELA code one of the rare codes capable of addressing the problem of edge-core turbulence. It is already capable of studying the impact of the edge on the turbulence in the core for electrostatic simulations (i.e. simulations where electrons are considered to be adiabatic). In order to run these simulations, it uses petascale high performance calculations (100 million CPU hours/year). The long-term objective for the code is to simulate a turbulent plasma in both the edge and the core with kinetic electrons for the international tokamak ITER which is currently being built at Cadarache (France). We already know that such simulations will not only require tomorrow's exascale resources, but also major numerical changes in the code.

This thesis lies within this context and it has a double objective: (i) develop new scalable numerical methods, adapted to the semi-Lagrangian scheme used in the GYSELA code, capable of solving the problem of large fluctuations and temperature variations (1-2 orders of magnitude) at the edge of the plasma, and (ii) take into account more realistic magnetic configurations than the concentric circles currently simulated by the code.

Concerning the handling of steep gradients at the edge of the plasma, as the current 5D grids (3 spatial coordinates, 2 velocity coordinates) already represent more than 100 billion points, the proposed solution for adding more points in the edge region is to use a non-equidistant mesh in the radial direction. Even if splines, which have up

till now shown themselves to be the best compromise in terms of calculation precision and cost, are retained, adding non-equidistant meshes requires in-depth changes to the compute kernels. From a theoretical perspective, I present a new approach for quadrature using splines, which limits the condition number for the procurement of such quadrature coefficients. One of the disadvantages of splines for their parallelisation is their global nature. I present a local spline method where derivatives are transported between patches, and show its stability for semi-Lagrangian advection. From a numerical perspective, bearing in mind that the GYSELA code is a code with more than 50 000 lines, based on a hybrid MPI/OpenMP parallelisation, and optimised for more than 100 000 cores, the choice was made to carry out in-depth studies of the semi-Lagrangian method based on non-uniform splines on a model with reduced dimensionality. The problem examined is a Vlasov-Poisson 1D-1V model, used for studies of the plasma sheath, which is a section of the plasma, which presents numerically troublesome steep gradients. The existing VOICE code (which is a mini version of GYSELA), designed to study such problems, has been modified and optimised on a GPU to operate on a non-uniform mesh. These improvements allowed simulations to be carried out which were previously unattainable, and allowed the validation of the semi-Lagrangian method on non-uniform splines.

As regards the new more realistic magnetic configurations in the GYSELA code, the choice was made to implement a Culham equilibrium. This equilibrium has the benefit of being based on analytical formulae, while also taking into account the important shaping parameters of an equilibrium plasma: elongation, triangularity, and Shafranov shift. Co-variant and contra-variant transformation matrices were derived and implemented in the code to allow the 5D Vlasov equations to take this geometry into account. The Poisson equation has up to now been solved numerically by projecting each 2D poloidal slice into Fourier space in the periodic poloidal direction, and by using second order finite differences in the radial direction. This solver has been replaced by a 2D finite elements solver based on splines(Zoni and Güçlü 2019) which was extracted from the SELALIB library (SeLaLib Development Team 2018). The inclusion of this new magnetic configuration has been successfully numerically validated on the linear benchmarks used for geodesic acoustic mode (GAM) studies. In parallel, a test platform for the 2D Poisson solver was developed in order to numerically compare this spline finite elements solver to two other multi-grid solvers: (i) a solver based on the AMReX library(al. 2019) which uses finite volumes on a uniform cartesian mesh with embedded boundaries, and (ii) a solver developed by the CERFACS which uses finite differences on a logical mesh(Martin J Kühn, Kruse, and Rüde 2022). The advantage of a solver using embedded boundaries would be to more easily handle configurations with an X-point geometry. As for the solver developed by CERFACS, its main advantage would be to potentially use a highly optimised 3D multi-grid solver in order to be able to simulate stellarator configurations with the GYSELA code.

Keywords: semi-Lagrangian scheme, B-splines, non-equidistant meshes, Vlasov-Poisson equations, finite elements, HPC plasma turbulence simulations

Résumé

Prédire les performances des plasmas de fusion en termes de facteur d'amplification, autrement dit le rapport de la puissance fusion sur la puissance injectée, est l'un des challenges cruciaux dans la physique des plasmas de fusion. Dans cette perspective, la turbulence et le transport de chaleur doivent être modélisés dans un cadre théorique précis consistant ici à utiliser des outils de simulations "premier-principes" non-linéaires. Les équations gyrocinétiques 5D (3 coordonnées d'espace, 2 coordonnées de vitesse) pour chaque espèce (ions & électrons), couplées aux équations 3D de Maxwell représentent une description auto-consistante appropriée du problème. Les études de transport au coeur des plasmas de tokamak ont maintenant atteint une maturité avec plusieurs codes premiers principes dans le monde capables d'aborder ce problème. Cependant, malgré leurs nombreux succès à ce jour, leur capacité de prédiction reste contrainte par le contenu énergétique en particulier dans le cas de décharges optimisées. Réussir à franchir ce cap demande de pousser les modèles gyrocinétiques vers la région de bord du tokamak et dans la mesure du possible de traiter sur un même pied d'égalité le transport de bord et de coeur.

Le code gyrocinétique 5D non-linéaire GYSELA (V. GRANDGIRARD, ABITEBOUL, J. BIGOT et al. 2016) développé à l'IRFM/CEA a la particularité d'être global (simulation de l'ensemble du tore), de ne pas faire d'approximation de séparation d'échelle ("code full F") et de forcer la turbulence via des sources de particules, de moment et de chaleur. Ajouté à cela une condition de frontière immergée imitant l'extraction de chaleur par un limiteur, le code GYSELA est l'un des rares codes au monde capable d'adresser ce problème de turbulence plasma couplée coeur-bord. Il est déjà en capacité d'étudier l'impact du bord sur la turbulence de coeur pour des simulations electrostatiques (i.e où les électrons sont considérés adiabatiques). Il utilise pour cela de manière intensive les moyens de calcul haute performance petascales (100 millions d'heures CPU/an). L'objectif à long terme pour le code est de simuler une turbulence plasma couplée coeur-bord avec des électrons cinétiques pour le tokamak international ITER actuellement en construction à Cadarache (France). On sait d'ores et déjà que de telles simulations nécessiteront non seulement les ressources exascales de demain mais aussi des évolutions numériques majeures du code.

Cette thèse s'inscrit dans ce cadre et son objectif est double : (i) développer des méthodes numériques innovantes adaptées au schéma semi-Lagrangien utilisé dans le code GYSELA, passant à l'échelle, capables de résoudre le problème de grande amplitude de fluctuations et de variation de température (1 à 2 ordres de grandeurs) au bord du plasma et (ii) prendre en compte des configurations magnétiques plus réalistes que les configurations magnétiques concentriques circulaires jusqu'alors simulées dans le code.

Concernant le traitement de fort gradients aux bords du plasma, les maillages 5D (3D en espace et 2D en vitesse) actuels représentent déjà plus de 100 milliards de points, la solution envisagée pour raffiner le bord est d'utiliser un maillage non-équidistant dans la direction radiale. Même en conservant une interpolation par splines, qui s'est avérée le meilleur compromis jusqu'à présent en termes de précision et de coût de calcul, le passage à un maillage non-équidistant nécessite une modification en profondeur des noyaux de calcul. Sur le plan théorique, nous présentons une nouvelle approche pour la quadrature par splines, qui limite le conditionnement pour l'obtention des coefficients de quadrature. L'un des inconvénients des splines en terme de parallélisation est leur caractère global. Nous présentons une approche splines locales avec transport des dérivées entre chaque patch ; et démontrons sa stabilité pour une advection semi-Lagrangienne. Sur le plan numérique sachant que le code GYSELA est un code de plus de 50 000 lignes, basé sur une parallélisation hybride MPI/OpenMP et optimisé à plus de 100 000 coeurs, le choix a été fait de réaliser les études approfondies des méthodes semi-Lagrangiennes basées sur des splines non-uniformes sur un modèle de dimensionnalité réduite. Le problème considéré est un modèle Vlasov-Poisson 1D-1V utilisé pour l'étude de la gaine dans un plasma qui présente de très forts gradients numériquement contraignants. Le code VOICE (mini-application de GYSELA) qui était utilisé jusqu'à présent pour les simulations de gaine a été modifié et optimisé sur GPU pour prendre en compte un maillage non-équidistant. Ces améliorations ont permis d'atteindre des simulations encore jamais réalisées et de valider l'approche semi-Lagrangienne avec splines non-uniformes.

Concernant la prise en compte d'une configuration magnétique plus réaliste dans le code GYSELA, le choix a été fait d'implémenter un équilibre de Culham. L'avantage de cet équilibre est de rester sur des bases analytiques, tout en permettant de prendre en compte les paramètres importants d'un équilibre plasma qui sont l'élongation, la triangularité et le décalage de Shafranov. Les matrices co-variantes et contra-variantes de transformations ont été dérivées et implémentées dans le code pour une prise en compte dans les équations de Vlasov 5D. Le solveur de Poisson – jusqu'à présent résolu numériquement en projetant chaque coupe poloidale 2D dans l'espace de Fourier dans la direction périodique poloidale et par des différences finies d'ordre 2 dans la direction radiale – a été remplacé par un solveur 2D éléments finis basés sur des splines (ZONI et GÜÇLÜ 2019) qui a été extrait de la librairie SELALIB(SELALIB DEVELOPMENT TEAM 2018). La prise en compte de cette configuration magnétique plus réaliste a été validée numériquement avec succès sur les benchmarks linéaires d'étude des modes géodésiques accoustiques (GAM). En parallèle, une plate-forme de tests du solveur de Poisson 2D a été développée pour pouvoir comparer numériquement ce solveur basé sur des éléments finis splines à deux autres solveurs multigrilles : (i) l'un basé sur la bibliothèque AMREX(AL. 2019) qui utilise des volumes finis sur un maillage cartésien uniforme et (ii) l'autre, développé par le CERFACS qui utilise des différences finies sur un maillage logique(Martin J KÜHN, KRUSE et RÜDE 2022). L'avantage d'un solveur basé sur des frontières immergées serait de pouvoir traiter plus facilement des configurations avec point de champ-X. L'avantage d'utiliser le solveur développé au CERFACS serait quant à lui de pouvoir bénéficier d'un solveur 3D multigrille très

optimisé en vue de pouvoir à terme simuler avec le code GYSELA des configurations stellarators.

Mots clés : schémas semi-lagrangiens, B-splines, maillages non-équidistants, équations Vlasov-Poisson, éléments finis, simulations HPC de plasma turbulent

Acknowledgements

I would like to begin by thanking my thesis supervisors Virginie Grandgirard and Michel Mehrenberger for their valuable advice throughout this thesis. Virginie was always there for any questions regarding GYSELA and the breadth of her knowledge is always impressive. For mathematical questions Michel's insight was invaluable.

I would also like to thank Yaman Güçlü for all his advice. His knowledge of splines was particularly helpful. In addition, his rigorous approach to problem solving has helped me out of tight coding corners on more than one occasion both during my thesis and during my master's project which he supervised.

I would like to thank all my colleagues at IRFM/CEA, including but not limited to Philippe Ghendrih, Yannick Sarazin, Xavier Garbet, Guilhem Dif-Pradalier, Peter Donnel. In particular I would like to thank Yann Munschy for all his help with the physics side of problems and his enthusiasm each time the code was accelerated. I would also like to thank Kevin Obrejan in particular for his help with technical queries concerning the quasi-neutrality equation.

I express my gratitude to the members of my jury, Bruno Després, Virginie Ehrlacher, Philippe Helluy, Carola Kruse, Claudia Negulescu, and Eric Sonnendrücker for their interest in this work.

I would also like to thank to my co-authors on the paper (Bourne, Leleux, Kormann, et al. 2022), Katharina Kormann, Martin J. Kühn, Ulrich Rüde, and Edoardo Zoni, and in particular Philippe Leleux for the time he spent debugging troublesome edge cases.

Finally I would like to thank my friends and family for their support, and in particular Olivier Kauffmann for helping provide the best possible conditions for writing this thesis.

Centre de Calcul Intensif d'Aix-Marseille is acknowledged for granting access to its high performance computing resources.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 800945 — NUMERICS — H2020-MSCA-COFUND-2017

Contents

List of Figures

List of Tables

List of acronyms

ASDEX

Axially Symmetric Divertor Experiment. 28

BGK

Bhatnagar-Gross-Krook. 76, 77, 83

CFL

Courant–Friedrichs–Lewy. 30, 114

DDC

discrete domain computation. 33

DEMO

Demonstration Power Plant. 28

EoCoE-II

Energy oriented Centre of Excellence 2. 34

FDM

Finite Difference Method. 30, 31, 33, 82, 131, 134

FEM

Finite Element Method. 30, 33, 82, 136, 187

FFT

Fast Fourier Transform. 31, 130, 134

FVM

Finite Volume Method. 30, 33

GAM

Geodesic Acoustic Mode. 33, 122, 138, 143, 187

GmgPolar

Geometric multigrid solver for curvilinear coordinates. 152–155, 158, 163–171, 178, 182

HPC

High Performance Computing. 29, 31

ITER

International Thermonuclear Experimental Reactor. 28

JET

Joint European Torus. 28

KEEN

Kinetic Electrostatic Electron Nonlinear. 72

LCFS

Last Closed Flux Surface. 73, 123, 175

MHD

Magnetohydrodynamics. 29

NSTX

National Spherical Torus Experiment. 175

PB

Petabytes. 32

PIC

Particle In Cell. 30, 71, 81, 114

SOL

Scrape-Off Layer. 73

TCV

Tokamak à Configuration Variable. 28

WEST

W Environment in Steady-state Tokamak. 28, 121

1. Introduction

Energy production is a critical problem in the world today. Population increases and technological advancement lead to ever increasing energy demands. Currently all methods of production have significant downsides. Coal, oil, and gas all create carbon emissions contributing to global warming. Wind and solar power do not release carbon emissions, but are both subject to favourable weather conditions and require large areas of land to generate useful quantities of energy. Meanwhile, hydroelectric power can only be harnessed in a select number of locations depending on the appropriate geography being available. Finally, nuclear power does not release carbon emissions, and is reasonably compact, but it generates highly radioactive waste material and accidents can occasionally have catastrophic effects.

One possible solution to this problem is to generate power via nuclear fusion. This reaction generates large quantities of energy from small quantities of a readily-available fuel: hydrogen. The process does not release carbon emissions, and requires external input in order to propagate. As a result it is not susceptible to dangerous accidents. Finally although the process generates small quantities of radioactive waste, this waste is minimal and has a half-life of hundreds of years, compared to several thousands for nuclear fission waste.

1.1. Nuclear Fusion

Nuclear fusion is the process through which the nuclei of two atoms fuse together to form one nucleus of a heavier element. This process liberates the additional binding energy necessary to hold the nuclei together. Figure 1.1 shows the binding energy as a function of the atomic mass of an element. We can see that this process only liberates energy when lighter elements are fused together to form heavier elements. Nuclear fission also produces energy by liberating binding energy, this time by splitting heavy elements into lighter elements, however as shown in Figure 1.1 this process releases significantly less energy than fusion reactions with a hydrogen fuel source.

In order to fuse elements, the nuclei must overcome electromagnetic forces to arrive sufficiently close to one another for strong forces to become dominant, allowing the creation of the fused nucleus. In order for the nuclei to overcome the electromagnetic forces, they must have sufficient energy. In practice this means that they must be sufficiently hot (more than one million degrees). At such high temperatures electrons are stripped off atoms leaving behind a soup of ions and electrons, known as a plasma.

The plasma must also be confined to ensure a sufficient density for nuclei to encounter one another. As a plasma is ionised this can be done using large powerful

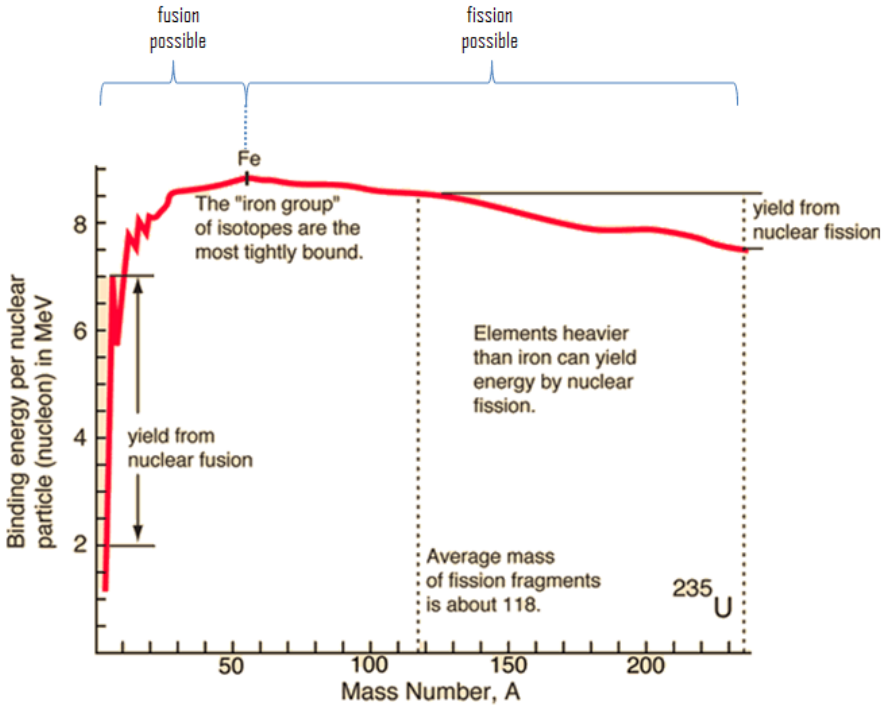


Figure 1.1.: Binding energy of different elements
Source: www.nuclear-power.com

magnets. There are two main categories of machines capable of creating these conditions: tokamaks which are axisymmetric, and stellarators which have a more complicated twisted geometry (Chen 2016). There are several existing tokamaks, including Joint European Torus (JET)[1] at Culham in the UK, W Environment in Steady-state Tokamak (WEST)[2] at IRFM/CEA in France, where I conducted my thesis, Tokamak à Configuration Variable (TCV)[3] at Lausanne in Switzerland, and Axially Symmetric Divertor Experiment (ASDEX)[4] at Garching in Germany. Future, larger tokamaks have already been planned including International Thermonuclear Experimental Reactor (ITER)[5] (first plasma planned for 2025), and Demonstration Power Plant (DEMO)[6] (first plasma planned for after 2050).

Although nuclear fusion is a promising domain, there are still several challenges before this technique can be employed in energy production. In order for fusion reactions to produce net power, the plasma must be maintained in a state where enough fusion occurs, for at least one second. If too little fusion occurs then the output power will not offset the power necessary to heat and confine the plasma (Gibney 2022). This milestone has not yet been achieved. The JET tokamak holds the

[1] https://ccfe.ukaea.uk/research/joint-european-torus/
[2] https://irfm.cea.fr/en/west/
[3] https://www.epfl.ch/research/domains/swiss-plasma-center/research/tcv/
[4] https://www.ipp.mpg.de/16195/asdex
[5] https://www.iter.org/
[6] https://www.euro-fusion.org/programme/demo/

current record for the highest sustained energy pulse (Gibney 2022).

In order to push the results further, to the point where fusion can become an energy source, we must develop a theoretical understanding of the processes that control plasma behaviour.

1.2. Plasma

A plasma is a state of matter where unbound ionised particles behave similarly to atoms in a gas. In tokamaks, this state is obtained by heating a gas to temperatures so high that electrons have enough energy to break free from the atoms.

Plasma behaviour at a microscopic scale is described by Maxwell's electromagnetic equations. At a macroscopic scale the equations describing the behaviour are not yet well defined. One reason for the difficulty in describing plasmas macroscopically is their inherently turbulent nature. Turbulence is particularly difficult to describe analytically so it is important to create reduced models which provide insights into how to control the plasma.

Additional knowledge in this domain can only be obtained by analysing experimental data. Nevertheless, physical experiments are extremely costly and the machines are not easily configurable once built. For example, although changes can be made to the internal shape of a machine, it is not cost-effective to do this more than once a decade. One way to access more varied experimental results without incurring astronomical costs is to use numerical experiments.

1.3. Numerical Plasma Simulations

Although the microscopic behaviour of a plasma can be described analytically using Maxwell's electromagnetic equations, a simulation modelling all 10^{22} particles (for a plasma in an ITER-sized tokamak), based only on these equations would be extremely costly. On today's High Performance Computing (HPC) machines we would therefore be limited to simulations covering a tiny area in space and only simulating a few nanoseconds; otherwise the simulation would run for a prohibitively long time and would not fit in the memory of the supercomputer.

Methods exist which reduce the size of the problem to allow results to be obtained in a reasonable time. The most simplified models are fluid models or Magnetohydrodynamics (MHD) models. In this case the plasma is modelled as a magnetised or non-magnetised fluid. As a result the simulation can rely on our extensive knowledge of fluid behaviour and only uses three dimensions. This results in relatively cheap simulations with around 10^7 points. However this change neglects the charges inside the plasma which have a non-negligible effect on the turbulence. Charges are also important in other domains such as plasma sheath physics, which describes the separation of charges near a wall. Jorek (Guido Huijsmans and Czarny 2007) is an example of a MHD code, while Soledge3X (H. Bufferand, Bucalossi, Ciraolo, et al. 2021) is an example of a fluid code.

A more common and more precise set of models are kinetic models. In this case the simplification that is made is that the particle distribution function of each species in the plasma is studied instead of the individual particles. The distribution function is a continuous function defined in both space and velocity space. Simulations describing these functions are therefore 6D.

Amongst the set of kinetic models, the gyrokinetic model is of particular interest (Xavier Garbet, Idomura, Laurent Villard, et al. 2010). In this case, the model is reduced further by neglecting the rapid gyration of the charged particles around the magnetic field lines. This gyration occurs at a frequency much higher than any other phenomena in the plasma, so this movement has negligible impact on the behaviour of the plasma. This simplification allows the problem to be reduced from six phase-space dimensions to five. Despite this reduction the resulting simulations still remain very large using around 10^{11} points. This thesis falls within the context of gyrokinetic models.

Several numerical methods exist to solve the gyrokinetic equations. They generally fall into two categories, Lagrangian methods and Eulerian methods.

Particle In Cell (PIC) methods are an example of a Lagrangian method. They model the distribution function with macro particles which are followed throughout the duration of the simulation. One disadvantage of the PIC method, and Lagrangian methods in general, is that the choice of macro particles can have an effect on the solution. Certain physical phenomena can push particles away from specific spatial regions. While some remain, if these particles were not in the original subset, the PIC method will have no information about the behaviour of the plasma in this region. Examples of EU codes using this method are the ORB5 code (S. Jolliet, Bottino, Angelino, et al. 2007) and EUTERPE (Hatzky, Tran, Könies, et al. 2002).

Eulerian methods such as Finite Difference Method (FDM), Finite Element Method (FEM), or Finite Volume Method (FVM) are used by codes such as GENE (Jenko, Dorland, Kotschenreuther, et al. 2000). A disadvantage of these methods is that they have Courant–Friedrichs–Lewy (CFL) conditions which restrict the size of the time step for explicit methods.

In addition to Lagrangian and Eulerian methods, the semi-Lagrangian method can also be used. This method was designed to eliminate the aforementioned disadvantages of the Lagrangian and Eulerian methods. This is done by evaluating the distribution function on a grid, as in Eulerian methods, but treating those grid points like macro particles during advections, as in Lagrangian methods. This technique will be described in detail in Chapters 3 and 4. The GYSELA code (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016) is a gyrokinetic semi-Lagrangian code, and it will be the focus of this thesis.

1.4. GYSELA

The GYSELA code takes its name from its main numerical methods: “Gyrokinetic Semi-Lagrangian”. It is a highly parallel 5D first principle code, which has been developed at the IRFM/CEA since 2001. Additional contributions are provided from researchers

and engineers from a plethora of different institutions including the Maison de la Simulation in Paris, IPP in Garching, Germany, the University of Strasbourg, and the University of Marseille. Such collaborations involving people from different research areas including HPC development, mathematics, and plasma physics, are essential to develop an efficient code, using state of the art numerical and physical methods.

The model used in the code is based on the gyrokinetic Vlasov-Poisson equations. The Vlasov equation, describing the movement of particles, contains a multitude of terms modelling not only the basic movement of particles in an electric field, but also collisions, source terms, and various sink terms, including penalisation sink terms describing the wall surrounding the plasma. This equation is solved using Strang splitting (Strang 1968). The resulting independent equations describing the movement are solved using the backwards semi-Lagrangian method which will be described in detail in Chapter 3. This method currently relies on uniform cubic splines. The additional terms each have dedicated solvers which have been discussed in other works (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016). Before this thesis, the Poisson equation was solved by projecting 2D poloidal slices into Fourier space in the poloidal direction using Fast Fourier Transform (FFT)s and using second order FDM radially. This choice will be discussed in Chapters 5 and 6.

The code is written in Fortran with some modules in C. It uses both OpenMP and MPI parallelism to run efficient simulations on more that 100 000 cores. Each simulation runs for several days, for a total of up to 100 million CPU hours per year. Each timestep generates 2TB of data. These simulations run on petascale supercomputers including the GENCI network of French national resources [7] (including IDRIS in Orsay, CINES in Montpellier and TGCC in Bruyères-Le-Châtel), the EU Fusion dedicated resource MARCONI[8], and the Japanese Fugaku computer[9]. These statistics put the code at the limits of current petascale resources. As GYSELA simulates larger and larger plasmas, aiming to simulate the large ITER tokamak, these requirements will be increased further requiring exascale HPC capacities.

Exascale machines are currently being planned and constructed, however these machines have a different architecture to their forebears. Specifically, they are based on heterogeneous accelerated computing nodes. This choice requires GYSELA to be performant on a variety of architectures, including GPUs and ARM instruction sets which have not been used in GYSELA until recently. First tests with ARM have unfortunately not shown promising results (loss of efficiency of a factor of 3 on the Fugaku machine). A rewriting of the code in C++ using modern programming models is therefore necessary.

[7] https://www.genci.fr/fr
[8] https://www.hpc.cineca.it/hardware/marconi
[9] https://www.fujitsu.com/global/about/innovation/fugaku/

1.5. Edge-Core Challenges

Since its inception, GYSELA has been capable of using gyrokinetic models to study turbulent plasma in the core of a tokamak (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016). However over time it has extended its capabilities to allow it to simulate a larger region extending up to the plasma wall (Guilhem Dif-Pradalier, Philippe Ghendrih, Yanick Sarazin, et al. 2022). This carries with it new challenges which require specially designed numerical methods.

Firstly, the edge region contains very steep gradients. Guilhem Dif-Pradalier, Philippe Ghendrih, Yanick Sarazin, et al. 2022 have shown that this region has an important effect on the turbulence in the core. It is therefore vital to ensure that this area is sufficiently discretised to productively model the gradients. Although this can be done by increasing the number of points uniformly, this would be wasteful as the additional refinement is not necessary in the core region. Furthermore the GYSELA code already runs simulations using 10 Petabytes (PB) of data. Increasing the memory excessively can therefore easily result in a simulation which is too large to run on today's petascale computing resources. This thesis will therefore focus on non-uniform methods as a solution to this problem.

Another problem as we extend the simulation towards the edge region is the breakdown of previously valid hypotheses. In the core region a circular cross section is a reasonable representation of the geometry. This is therefore the geometry that has thus far been used in the GYSELA code. However in the edge region this geometry provides a poor representation of the shape of the cross-section. This thesis will also examine the implementation of a new, more realistic geometry known as the "Culham" geometry and the tools necessary to handle this change.

1.6. Outline of the thesis

In Chapter 2 a review of splines, which underpin the majority of the methods used in this thesis, is provided. Additionally splines as tools for quadrature are discussed and I propose a new method for calculating the quadrature coefficients of Schoenberg's "best" quadrature, with significantly improved precision.

These splines are used in Chapter 3 to improve an existing code, known as VOICE (Vlasov Open boundary Ion Coupling to Electrons), which examines sheath physics. This problem demonstrates many of the problems due to steep gradients that are also encountered by GYSELA, while remaining simpler as it is only 1D-1V. Furthermore the VOICE code is written using many of the same numerical tools as the GYSELA code. It is therefore used as a test bed to trial non-uniform splines and examine the performance costs incurred by abandoning uniform splines. In (Bourne, Munschy, Virginie Grandgirard, et al. 2022), the advantages are shown to outweigh the costs in this case, and I further use GPUs to tip the scales in favour of the non-uniform solution. The resulting simulation can model more complex problems allowing a better understanding of the sheath, as presented by Munschy, Bourne, Guilhem Dif-

Pradalier, et al. 2022.

As a mini version of GYSELA, the VOICE code provides a perfect test bed to trial the use of modern C++ to create a more performant version of the code. As part of my thesis, I contributed towards the development of the new code VOICE++. I provided the spline modules, which were previously tested in VOICE by coupling Fortran and C++ code using Iso C Bindings. These modules were modified with the help of researchers from the Maison de la Simulation, to use a new optimised discrete domain computation (DDC) (Julien Bigot and Padioleau 2021) library.

In Chapter 4 I present a new approach to local splines as a solution to avoid the costs incurred due to non-uniform points by using locally uniform splines. As an additional benefit this approach would allow GYSELA to further increase its parallelism or avoid some of the costly MPI communications required to ensure that all data associated with a given dimension is available on one node for the construction of a global spline. Moreover, I prove the stability of the semi-Lagrangian method on these local splines. The next step for this work will be to implement this method in VOICE++.

To handle a more realistic geometry, the GYSELA team made the choice to use an "analytical" magnetic equilibrium to define the coordinates, rather than coupling the code with an equilibrium code such as CHEASE (Lütjens, Bondeson, and Sauter 1996) or CEDRES (Hertout, Boulbe, Nardon, et al. 2011). In Chapter 5, this new "analytical" equilibrium, referred to as the "Culham" equilibrium, is introduced. The changes necessary to add this geometry to the GYSELA code are discussed. This includes a discussion on generalised coordinates and the coupling of a new solver, extracted from the SeLaLib library (SeLaLib Development Team 2018), for the quasi-neutrality equation. This 2D FEM solver includes a special treatment of the singular point arising in the curvilinear coordinates, using $\mathcal{C}^1$ polar splines (Zoni and Güçlü 2019). Results of Geodesic Acoustic Mode (GAM) tests using the GYSELA code with this new geometry are compared to results from other major codes to show the successful implementation. This work was carried out in collaboration with Kevin Obrejan, Peter Donnel and two interns (Ken Leleux and Baptiste Legouix).

Finally the chosen solver, is compared with two alternative solvers: (i) a FDM solver using a multigrid method to solve the problem on a curvilinear mesh, developed by CERFACS (Martin J Kühn, Kruse, and Rüde 2022), and (ii) a FVM solver using a multigrid method to solve the problem on a cartesian mesh with embedded boundary conditions, developed using the AMReX library (al. 2019). In order to facilitate this comparison I developed a python script which auto-generated the code necessary to describe the various problems investigated, for each of the three solvers, in Fortran and C++, with the help of the pyccel library (Bourne, Güçlü, Saïd Hadjout, et al. 2022). This open-source library is a transpiler which translates python code to Fortran or C. It featured heavily in my master's thesis (Bourne 2018) and I have since continued to contribute to its development. This comparison is used to determine which solver provides an implementation which is best suited to the restrictions imposed by the code and the supercomputers where GYSELA is run. Advantages and disadvantages are shown for each of the three solvers in (Bourne, Leleux, Kormann, et al. 2022). Additionally I examine the difficulties associated with using the solvers on more

complex geometry such as an X-point geometry. This work was carried out as part of an Energy oriented Centre of Excellence 2 (EoCoE-II) project in collaboration with the Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique (CERFACS) and Max Planck Institute for Plasma Physics (IPP Garching).

Finally, in Chapter 7 the results of the thesis are summarised.

2. Splines

2.1. Introduction

Spline-based tools are used throughout this thesis for plasma simulations. In order to evaluate these methods, it is important to have a thorough understanding of what a spline is, and how it is used in different contexts. Splines have already been explained in detail by several different authors (Piegl and Tiller 1996; Farin 1993; Gordon and Riesenfeld 1974; De Boor 1978; de Boor 1972; Hämmerlin and Hoffman 1991; Patrikalakis and Maekawa 2002), but this chapter aims to summarise this work and present the sections relevant to this thesis.

In Section 2.2, the theory underpinning splines is explained in general terms. In practice, most numerical methods to not remain this general, but are instead expressed on the spline basis functions. These functions and their derivation are explained in Section 2.3. In Section 2.4, I explain how this framework can be extended to handle multi-dimensional problems, including polar domains using the method proposed by Toshniwal, Speleers, Hiemstra, et al. 2017. Next, in Section 2.5, I explain how splines are constructed in the context of interpolation problems. The most common boundary conditions for this problem are presented in section 2.5.1. In Section 2.6, I explain how the splines are evaluated and discuss the costs associated with solving general problems vs highly specialised problems. Following this I explain how the derivatives are calculated in Section 2.7. Finally I discuss how to integrate spline representations in Section 2.8.1. Spline integration can be leveraged to provide precise quadrature schemes. This is discussed in Sections 2.8.1 and 2.8.2. In particular Section 2.8.2 discusses the "best" quadrature using Schoenberg's criteria. This theoretical result is not often used as the existing methods for calculating the quadrature coefficients are poorly conditioned. In addition to presenting the existing methods, I also present my new method for calculating the coefficients which presents significantly better conditioning. Having avoided the conditioning problems which prevented the use of this quadrature, I present comparisons of this method with classic Newton-Cotes quadrature.

2.2. General Theory

Splines are smooth piecewise polynomial functions defined over a given domain $[a,b]$. For the rest of this work they will be noted by $S_d(x)$ where d is the maximum degree of the polynomials. They are characterised by n_c+1 break points $a = z_0 < z_1 < \cdots < z_{n_c-1} < z_{n_c} = b$ which split the domain $[a,b]$ into n_c cells. The n_c pieces of the spline

are noted $S_{j,d}(x)$ and are each polynomials of degree d or less, which can be described as follows:

$$S_{j,d}(x) = \sum_{i=0}^{d} p_{d,i,j} x^i \qquad \forall 0 \leq j < n_c \tag{2.1}$$

where $p_{d,i,j}$ are the polynomial coefficients for the j-th polynomial of degree d or less. The spline is defined as:

$$S_d(x) = \begin{cases} S_{0,d}(x) & \text{if } x \in [z_0, z_1[\\ \vdots & \\ S_{n_c-1,d}(x) & \text{if } x \in [z_{n_c-1}, z_{n_c}[\end{cases} \tag{2.2}$$

The most general spline is therefore characterised by $n_c(d+1)$ unknowns.

Different conditions of smoothness can be imposed on the splines at the break points $\{z_j\}$. The smoothest and most common splines are constructed such that $S_d(x) \in \mathcal{C}^{d-1}([a,b])$ (where $d \geq 1$, splines of degree 0 are discontinuous). Unless otherwise specified, in this work splines will always respect this condition. In other words, the splines and their first $d-1$ derivatives are continuous. This condition can also be expressed as:

$$\partial_x^k S_{j,d}(z_{j+1}) = \partial_x^k S^{(k)}_{j+1,d}(z_{j+1}) \qquad \forall 0 \leq k < d \quad \forall 0 \leq j < n_c - 1 \tag{2.3}$$

The condition adds $d \cdot (n_c - 1)$ constraints to the system which reduces the number of degrees of freedom of the system from $n_c \cdot (d+1)$ to

$$n_c \cdot (d+1) - d \cdot (n_c - 1) = n_c d + n_c - n_c d + d = n_c + d.$$

2.3. B-Splines

There are multiple ways to describe splines. The most straightforward definition describes the piecewise polynomials using the polynomial coefficients as described in equation (2.1). However this description has $n_c \cdot (d+1)$ unknowns. When handling smooth splines, there are only $n_c + d$ degrees of freedom, it is therefore possible to find a simpler description which takes the smoothness into account and only requires $n_c + d$ unknowns. This description is obtained by expressing the spline on basis splines, or b-splines:

$$S_d(x) = \sum_{i=0}^{n_c+d-1} c_i b_{i,d}(x) \tag{2.4}$$

where c_i are the coefficients of the spline which represent the degrees of freedom.

For a set of functions $\{b_{i,d}(x)\}$ to define a basis of the spline function space, they must respect the following conditions:

1. $b_{i,d}(x)$ is a piecewise polynomial function of degree $d \geq 0$ or less

2. $b_{i,d}(x) \in \mathcal{C}^{d-1}([a,b]), \quad \forall d > 0$

3. The set of all b-splines with support in the domain $[z_j, z_{j+1}]$ must define a basis of all polynomials of degree d or less

In addition, we require that the basis splines $b_{i,d}(x)$ have compact support.

As a polynomial of degree d has $d+1$ degrees of freedom, condition 3 implies that there must be exactly $d+1$ basis functions with support on an interval $[z_j, z_{j+1}]$. In order for this condition to be respected over the entire domain, while respecting the compact support condition, the support for each basis spline must cover at most $d+1$ consecutive intervals. This implies that it also has minimal support. We will note the support of the basis spline $b_{i,d}(x)$ as $[k_i, k_{i+d+1}]$ where k_i are known as the knots of the b-spline. Without loss of generality we assume that the k_i are ordered:

$$k_0 \leq k_1 \leq \cdots \leq k_{n_c+2d-1} \leq k_{n_c+2d} \tag{2.5}$$

Additionally as condition 2 is valid over the entire domain and not just the support of the function, we can also infer the following:

$$\partial_x^k b_{i,d}(k_i) = \partial_x^k b_{i,d}(k_{i+d+1}) = 0 \qquad \forall 0 \leq k < d \tag{2.6}$$

As each b-spline is a piecewise-polynomial of degree d defined over $d+1$ intervals its representation following equation (2.1) contains $(d+1)^2$ unknowns. The continuity condition 2 described by equations (2.3) and (2.6) imposes $d(d+2)$ conditions. These conditions are therefore not sufficient to define a unique set of basis splines. A final normalisation condition must be added to have a complete definition. The following normalisation choices are the most common:

Partition of Unity

$$\sum_{i=0}^{n_c+d} b_{i,d}(x) = 1 \qquad \forall x \in [a,b] \tag{2.7}$$

Unit Integral

$$\int_a^b Q_{i,d}(x)\,dx = 1 \tag{2.8}$$

where $Q_{i,d}(x)$ is the b-spline normalised to unit integral.

$Q_{i,d}(x)$ and $b_{i,d}(x)$ are related by the following equation (Boor, Lyche, and Schumaker 1976):

$$b_{i,d}(x) = \frac{\text{supp}(b_{i,d}(x))}{d+1} Q_{i,d}(x) = \frac{k_{i+d+1} - k_i}{d+1} Q_{i,d}(x) \tag{2.9}$$

As the support of $b_{i,d}(x)$ covers $d+1$ cells, in the case of equidistant knots this equation can be written as:

$$b_{i,d}(x) = \frac{h(d+1)}{d+1} Q_{i,d}(x) = h Q_{i,d}(x) \tag{2.10}$$

where $h = z_{j+1} - z_j$ is the spatial resolution.

In this work partition of unity is used for the normalisation. This gives us the following definition for the b-splines (Gordon and Riesenfeld 1974):

$$\begin{aligned} b_{i,0}(x) &= \begin{cases} 1 & \text{if } k_i \leq x < k_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ b_{i,d}(x) &= \frac{x - k_i}{k_{i+d} - k_i} b_{i,d-1}(x) + \frac{k_{i+d+1} - x}{k_{i+d+1} - k_{i+1}} b_{i+1,d-1}(x) \end{aligned} \tag{2.11}$$

Up to this point we have simplified the problem by ignoring the behaviour of the b-splines at the boundaries. For a smooth spline the break points can be mapped to the $n_c + 1$ inner knots:

$$k_{i+d} = z_i \qquad \forall\, 0 \leq i \leq n_c \tag{2.12}$$

However for splines of degree $d > 0$ a definition of the boundary knots is also required to fully define the b-splines in this region. These knots can be chosen arbitrarily, however a simple choice is:

$$\begin{cases} k_i = a - \frac{b-a}{n_c}(i-d) & 0 \leq i < d \\ k_{n_c+d+i} = b + \frac{b-a}{n_c} i & 0 \leq i < d \end{cases} \tag{2.13}$$

In the case of equidistant break points this adds d ghost cells to the boundary region, each having the same size as the cells within the domain. Figure 2.1 shows the b-splines for the domain $[0,3]$ defined using this choice of additional knots.

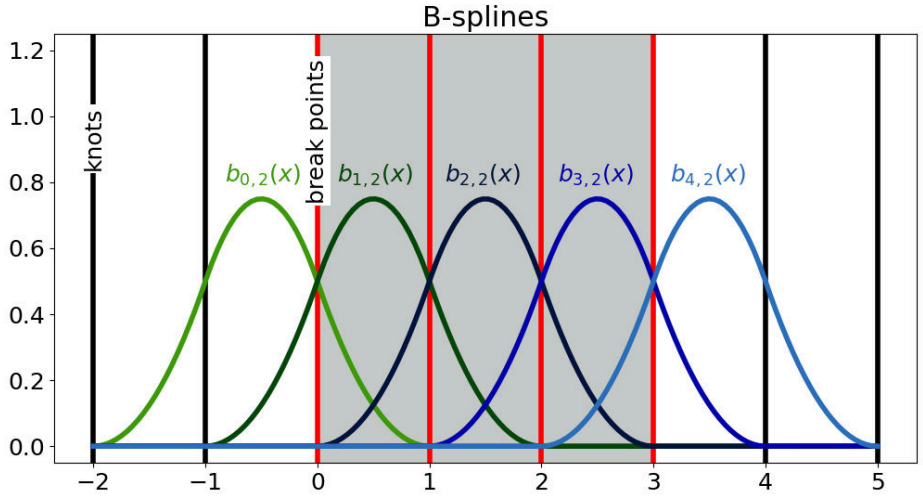


Figure 2.1.: B-splines of degree 2 for the domain $[0,3]$ with additional knots outside the domain

2.3.1. General B-Splines

B-splines can be used not only in the case where the continuity constraint $S_d(x) \in \mathcal{C}^{k-1}([a,b])$ is fully respected, but also in cases where the continuity condition is reduced at specific break points. In order to reduce the continuity at a break point to $\mathcal{C}^{k-1-r}$, r additional knots are placed at this point.

In this case the b-splines are more accurately described by the following equation:

$$\begin{aligned} b_{i,0}(x) &= \begin{cases} 1 & \text{if } k_i \leq x < k_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ b_{i,d}(x) &= \frac{x-k_i}{k_{i+d}-k_i} b_{i,d-1}(x)\delta_{k_{i+d}\neq k_i} + \frac{k_{i+d+1}-x}{k_{i+d+1}-k_{i+1}} b_{i+1,d-1}(x)\delta_{k_{i+d+1}\neq k_{i+1}} \end{aligned} \tag{2.14}$$

where δ_w is a function equal to 1 when w is true and 0 otherwise.

This definition leads to a new way of defining the additional knots at the boundary. Instead of placing equidistant points as described in equation (2.13) we may choose to place all additional knots on the boundary:

$$\begin{cases} k_i & = a & \forall 0 \leq i < d \\ k_{n_c+d+i} & = b & \forall 0 \leq i < d \end{cases} \tag{2.15}$$

This choice leads to the following properties for b-splines near the boundary:

$$\frac{\partial^j}{\partial x^j} b_{i,d}(x) = 0 \qquad \forall\, j+i < d \tag{2.16}$$

$$\frac{\partial^j}{\partial x^j} b_{n_b-i,d}(x) = 0 \qquad \forall\, j < i \tag{2.17}$$

where $n_b = n_c + d$ is the number of basis functions. Figure 2.2 shows the b-splines for the domain $[0,3]$ defined using this choice of additional knots.

This choice is especially useful for imposing boundary conditions when interpolating a function as we will see in section 2.5.

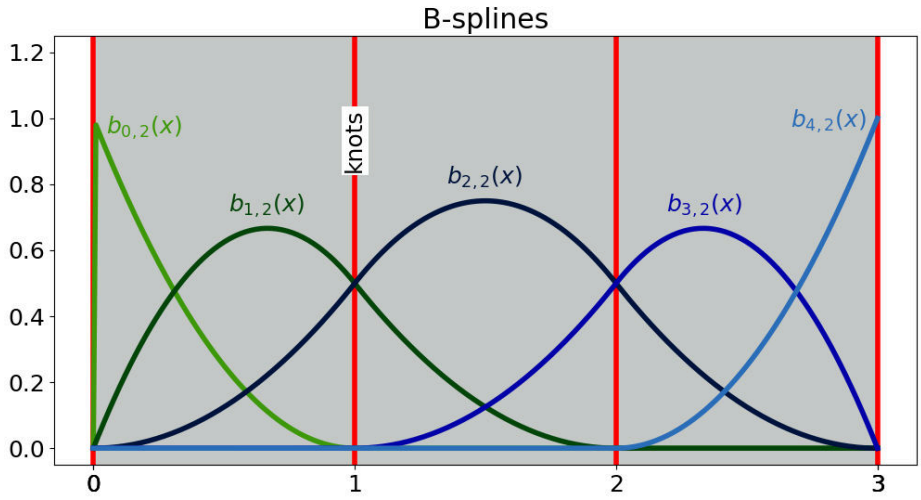


Figure 2.2.: B-splines of degree 2 for the domain [0,3] with additional knots at the edge of the domain

2.4. N-D Splines

Splines as described so far are inherently 1D objects, however this framework can easily be expanded to multiple dimensions. A spline defined on N dimensions is expressed on a b-spline basis as follows:

$$S_d(x_1, x_2, \ldots, x_N) = \sum_{j_1=0}^{n_{b1}} \ldots \sum_{j_N=0}^{n_{bN}} \prod_{i=1}^{N} b_{j_i,d}(x_i) c_{j_1,\ldots,j_N} \tag{2.18}$$

The basis of such a spline is therefore comprised of products of 1D b-splines.

As each dimension behaves independently, different degrees may be used along the different dimensions.

For example, the basis of a 2-D spline would be described as follows:

$$b_{i,j,d_x,d_y}(x,y) = b_{i,d_x}(x) b_{j,d_y}(y) \tag{2.19}$$

2.4.1. Polar coordinate systems

N-D splines are simple in the case of cartesian coordinates, but they can also be used with more complex geometry. Nevertheless, complications occur when trying to use splines with coordinate systems which have a singular point. While the behaviour is well defined away from the singularity, at the singularity there are many knots in the same place. This is not reflected in the spline representation however, which can lead to an undefined value at the singular point. Toshniwal, Speleers, Hiemstra, et al. 2017 have suggested a possible solution to this problem.

Their solution uses two tools which will be explained briefly before explaining the method. These tools are barycentric coordinates and Bernstein polynomials.

2.4.1.1. Barycentric Coordinates

Barycentric coordinates are a coordinate system used to locate a point $\vec{P}$ inside a triangle with vertices $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$. The barycentric coordinates, denoted $(\lambda_1, \lambda_2, \lambda_3)$, are the weights such that:

$$\vec{P} = \lambda_1 \vec{v}_1 + \lambda_2 \vec{v}_2 + \lambda_3 \vec{v}_3 \tag{2.20}$$

and $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

These weights have a physical interpretation. If point masses of weight λ_i are placed at the vertices $\vec{v}_i$, then the centre of mass can be found at the point $\vec{P}$. The coordinate values λ_i are defined on the domain $[0, 1]$ as long as $\vec{P}$ is located inside the triangle.

The coordinates are related to cartesian coordinates via the following equations:

$$\lambda_1 = \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)} \tag{2.21}$$

$$\lambda_2 = \frac{(y - y_3)(x_1 - x_3) + (x_3 - x)(y_1 - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)} \tag{2.22}$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2 \tag{2.23}$$

2.4.1.2. Bernstein Polynomials

Bernstein polynomials provide a basis for bivariate polynomials of degree k. A bivariate polynomial $p(x, y)$ of degree k can be written as:

$$p(x, y) = \sum_{i=0}^{k} \sum_{j=0}^{k-i} \alpha_{i,j} x^i y^j \tag{2.24}$$

where $\alpha_{i,j}$ are the coefficients of the polynomial. The number of basis functions n_b required to describe the system, is equal to the number of coefficients $\alpha_{i,j}$:

$$n_b = \sum_{i=0}^{k} \sum_{j=0}^{k-i} 1 = \sum_{i=0}^{k} (k + 1 - i) = \sum_{m=1}^{k+1} m = \frac{(k+1)(k+2)}{2} \tag{2.25}$$

Therefore $\frac{(k+1)(k+2)}{2}$ Bernstein polynomials are required to describe the space of bivariate polynomials of degree k.

These polynomials, denoted $T_{i_1,i_2,i_3}(\lambda_1, \lambda_2, \lambda_3)$, are defined as follows:

$$T_{i_1,i_2,i_3}(\lambda_1, \lambda_2, \lambda_3) = \frac{k!}{i_1! i_2! i_3!} \lambda_1^{i_1} \lambda_2^{i_2} \lambda_3^{i_3} \tag{2.26}$$

where $(\lambda_1, \lambda_2, \lambda_3)$ are the barycentric coordinates, and the indices i_1, i_2, i_3 are positive

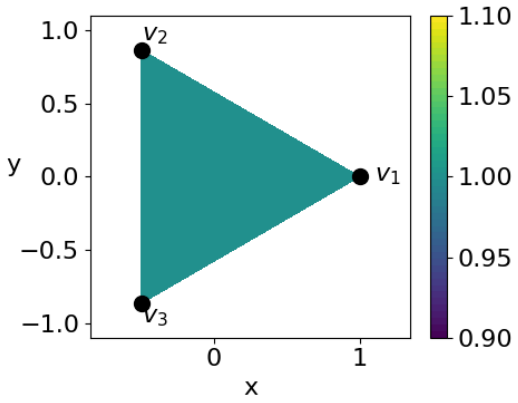


Figure 2.3.: Bernstein polynomial basis for zero degree bivariate polynomials

integers such that $i_1 + i_2 + i_3 = k$. In other words the indices are such that:

$$i_1 \in [0, k] \tag{2.27}$$
$$i_2 \in [0, k - i_1] \tag{2.28}$$
$$i_3 = k - i_1 - i_2 \tag{2.29}$$

Figures 2.3 - 2.5 show the shape of the Bernstein polynomials for degree zero, one, and two.

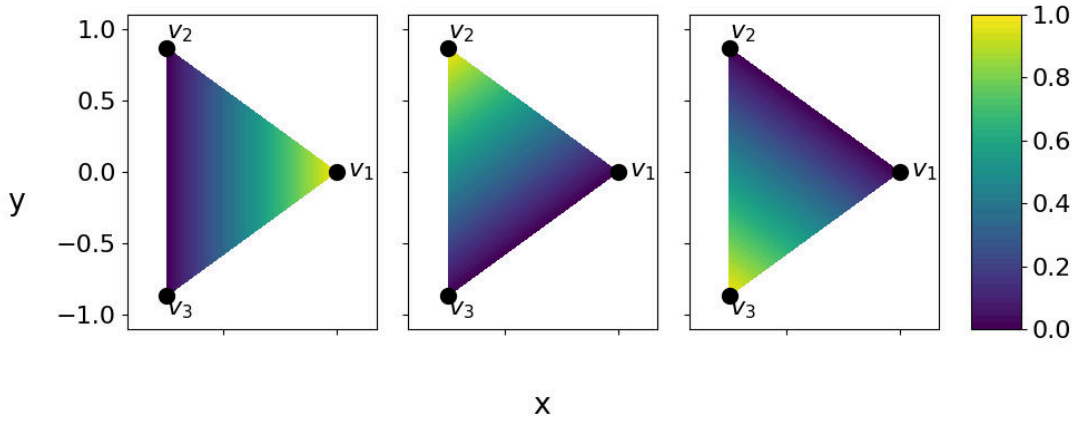


Figure 2.4.: Bernstein polynomial basis for first degree bivariate polynomials

Each of the Bernstein polynomials are positive on their domain triangle, and the basis respects partition of unity:

$$\sum_{i_1=0}^{k} \sum_{i_2=0}^{k-i_1} T_{i_1,i_2,k-i_1-i_2}(\lambda_1, \lambda_2, \lambda_3) = 1 \tag{2.30}$$

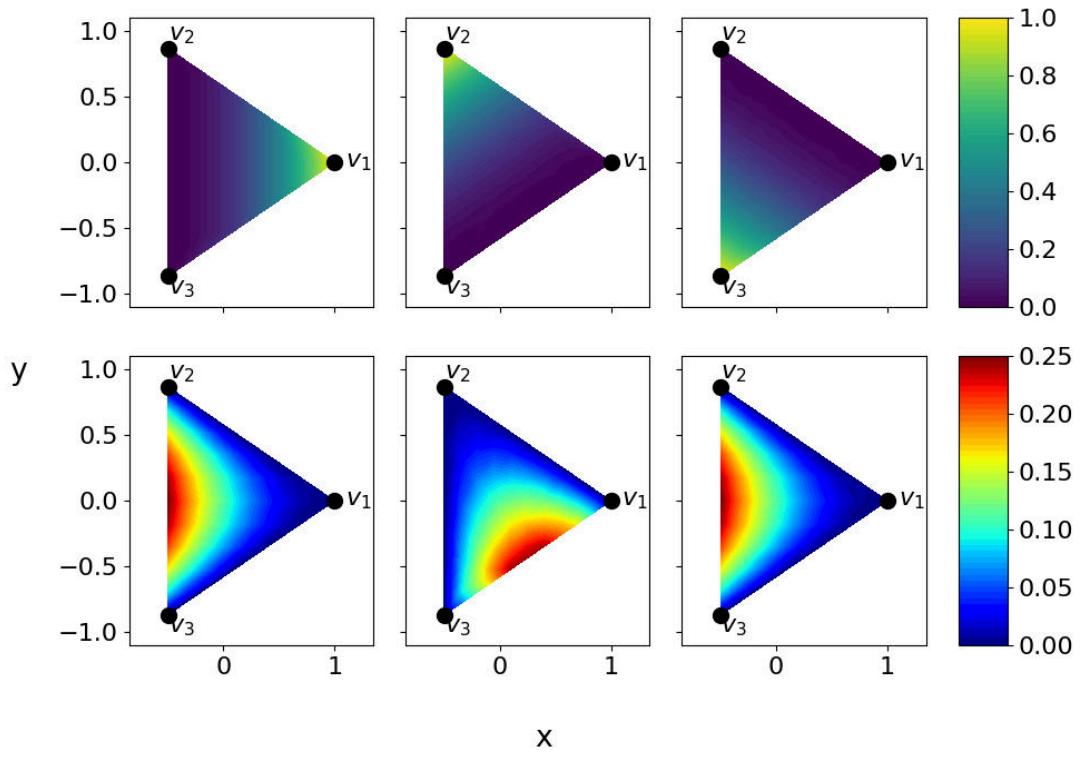


Figure 2.5.: Bernstein polynomial basis for second degree bivariate polynomials

2.4.1.3. $\mathcal{C}^k$ Polar Splines

In this section I will explain the method presented by Toshniwal, Speleers, Hiemstra, et al. 2017 for constructing $\mathcal{C}^k$ polar splines. The idea is based on replacing the innermost rings of the 2D b-splines defined on polar coordinates (r,θ), with linear combinations of these values. The innermost m b-splines are the b-splines:

$$b_{i,j,d_r,d_\theta}(r,\theta) \quad \forall 0 \leq i < m \quad \forall 0 \leq j < n_{b\theta} \tag{2.31}$$

Their physical interpretation can be seen in Figure 2.6.

In order to obtain a spline which is $\mathcal{C}^k$, the system must be constructed such that the value and the first k derivatives at the singular point are well-defined. Therefore the new basis splines must be linear combinations of all the old basis splines whose value or first k-th derivatives are non-zero at the singular point boundary. Assuming that all boundary knots are placed at the boundary and we are therefore in the configuration described in Figure 2.2, this means that we must replace the k+1 innermost rings.

For example, for a polar coordinate system, the b-splines which lead to a discontinuous representation in the centre, and must therefore be replaced, are defined as:

$$b_{i,j,d_r,d_\theta}(r,\theta) = b_{i,d_r}(r)\, b_{j,d_\theta}(\theta) \qquad \forall 0 \leq i < k, \quad \forall 0 \leq j < n_{b\theta} \tag{2.32}$$

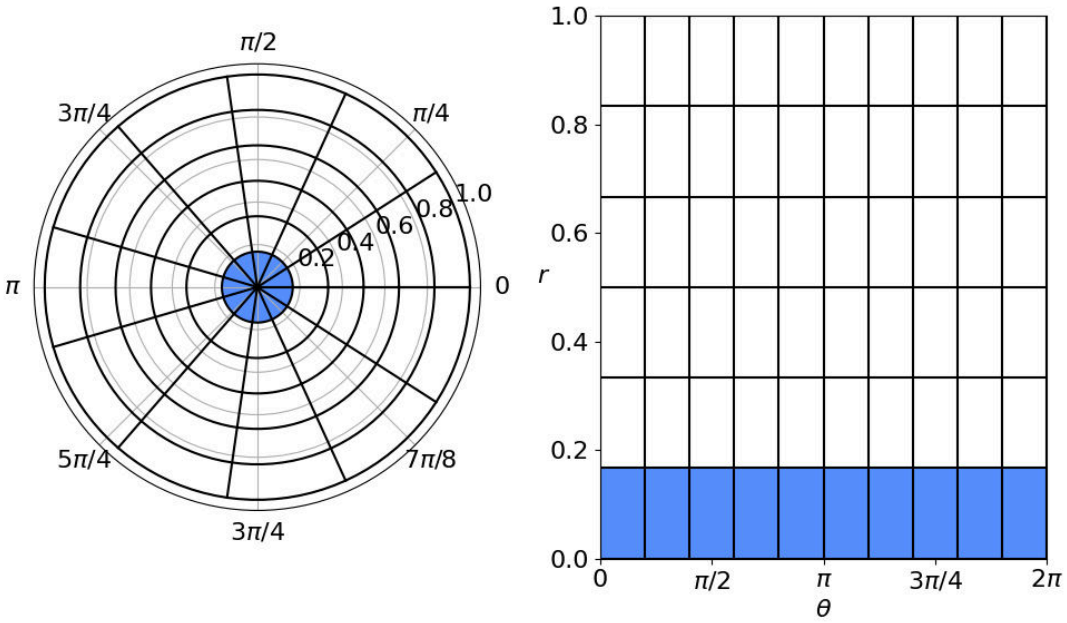


Figure 2.6.: The representation of the polar space on b-splines. The innermost ring which leads to a discontinuous representation at the singular point is highlighted in blue

Additionally, in order to obtain a spline which is $\mathcal{C}^k$ it is important that the basis functions over the singular point represent a basis of k-th degree bivariate polynomials. As explained in section 2.4.1.2 such a basis contains $\frac{(k+1)(k+2)}{2}$ elements. There are therefore $\frac{(k+1)(k+2)}{2} n_{b\theta} k$ coefficients to be determined in order to define the linear combinations.

In order to be sure that there are the correct number of degrees of freedom at the singular point, Toshniwal, Speleers, Hiemstra, et al. 2017 suggest that each of the new $\frac{(k+1)(k+2)}{2}$ basis functions $\hat{b}_l(r,\theta)$ are chosen such that they respect the following $\frac{(k+1)(k+2)}{2}$ conditions:

$$\lim_{(r,\theta)\to(0,0)} \frac{\partial^{m_1+m_2}}{\partial r^{m_1}\partial\theta^{m_2}} \hat{b}_l(r,\theta) = \frac{\partial^{m_1+m_2}}{\partial r^{m_1}\partial\theta^{m_2}} T_l(0,0) \tag{2.33}$$

where $m_1, m_2 \geq 0$ are integers such that $m_1 + m_2 \leq k$, and $T_l(r,\theta)$ is the l-th Bernstein polynomial, as defined in section 2.4.1.2.

The linear combination can be written as follows:

$$\begin{pmatrix} \hat{b}_0(r,\theta) \\ \hat{b}_1(r,\theta) \\ \vdots \\ \hat{b}_{\frac{1}{2}(k+1)(k+2)-1}(r,\theta) \end{pmatrix} = \begin{pmatrix} A_0 & A_1 & \dots & A_k \end{pmatrix} \begin{pmatrix} b_0^* \\ b_1^* \\ \vdots \\ b_k^* \end{pmatrix} \tag{2.34}$$

where:

$$b_i^* = \begin{pmatrix} b_{i,d_r}(r)b_{0,d_\theta}(\theta) \\ b_{i,d_r}(r)b_{1,d_\theta}(\theta) \\ \vdots \\ b_{i,d_r}(r)b_{n_\theta-1,d_\theta}(\theta) \end{pmatrix} \tag{2.35}$$

Thanks to the partition of unity of the original b-splines $b_{i,j,d_r,d_\theta}(r,\theta)$, and the choice of boundary knots we know that A_0 can be defined as follows:

$$A_0 = \begin{pmatrix} T_0(0,0) & T_0(0,0) & \dots & T_0(0,0) \\ T_1(0,0) & T_1(0,0) & \dots & T_1(0,0) \\ & \vdots & & \\ T_{\frac{1}{2}(k+1)(k+2)-1}(0,0) & T_{\frac{1}{2}(k+1)(k+2)-1}(0,0) & \dots & T_{\frac{1}{2}(k+1)(k+2)-1}(0,0) \end{pmatrix} \tag{2.36}$$

The remaining conditions relate to the derivatives. Taking the example of the first order derivative, we see that we have two equations to satisfy:

$$\lim_{(r,\theta)\to(0,0)} \frac{\partial}{\partial x}\hat{b}_l(r,\theta) = \frac{\partial}{\partial x}T_l(0,0) \tag{2.37}$$

$$\lim_{(r,\theta)\to(0,0)} \frac{\partial}{\partial y}\hat{b}_l(r,\theta) = \frac{\partial}{\partial y}T_l(0,0) \tag{2.38}$$

In polar coordinates this condition is written:

$$\lim_{(r,\theta)\to(0,0)} \frac{\partial}{\partial r}\hat{b}_l(r,\theta) = \frac{\partial}{\partial r}T_l(0,0) \tag{2.39}$$

$$\lim_{(r,\theta)\to(0,0)} \frac{\partial}{\partial \theta}\hat{b}_l(r,\theta) = 0 \tag{2.40}$$

Equation (2.40) must hold for the b-spline to be $\mathcal{C}^k$ (with k>0 such that we care about the first derivative) and is therefore already enforced by writing the b-spline as a linear combination.

2.4.1.4. $\mathcal{C}^1$ polar splines

In the case of $\mathcal{C}^1$ polar splines, equation (2.34) is written as:

$$\begin{pmatrix} \hat{b}_0(r,\theta) \\ \hat{b}_1(r,\theta) \\ \hat{b}_2(r,\theta) \end{pmatrix} = \begin{pmatrix} A_0 & A_1 \end{pmatrix} \begin{pmatrix} b_0^* \\ b_1^* \end{pmatrix} \tag{2.41}$$

Toshniwal, Speleers, Hiemstra, et al. 2017 have shown that the matrix $\begin{pmatrix} A_0 & A_1 \end{pmatrix}$ can be defined as:

$$A_{l,(i,j)} = T_l(k_{ij}) \tag{2.42}$$

where l is the row index, ij is the column index, corresponding to the indexation of

the original b-splines $b_{i,j,d_r,d_\theta}(r,\theta)$, and k_{ij} is the control point at the intersection of the knots r_i and θ_j. At $i=0$, $r_i=0$ which means that we correctly recover equation (2.36).

The new b-splines are therefore defined as:

$$\hat{b}_l(r,\theta) = \sum_{i=0}^{1}\sum_{j=0}^{n_\theta} T_l(k_{ij})\, b_{i,d_r}(r)\, b_{j,d_\theta}(\theta) \quad \forall 0 \le l < 3 \tag{2.43}$$

$$\hat{b}_{l+3}(r,\theta) = b_{m,d_r}(r)\, b_{p,d_\theta}(\theta) \quad \forall 0 \le l < (n_r-2)\, n_\theta \tag{2.44}$$

where $m = \lfloor l/n_{b\theta} \rfloor$ and $p = (l \mod n_\theta) + 2$.

This definition will be revisited in Chapter 6 where it will be used for a finite elements scheme.

2.5. Interpolating Splines

Splines are often used as a solution to interpolation problems. An interpolation problem consists of finding a function $\phi(x)$ which satisfies the following equation:

$$\phi(x_j) = f_j = f(x_j) \qquad \forall 0 \le j < n_p \tag{2.45}$$

for n_p given nodes: $x_0 < x_1 < \cdots < x_{n_p-1}$ on a domain $[a,b]$, and the corresponding values at those nodes $f_0 = f(x_0), ..., f_{n_p-1} = f(x_{n_p-1}) \in \mathbb{R}$, where $f(x)$ is the function which is interpolated.

In order to use a spline as a solution to the interpolation problem, knots must be chosen. One frequently used solution to this problem is to choose the break points z_i such that $z_i = x_i$. Figure 2.7 shows the interpolation of a Gaussian function by splines of degree 1 and 3 with break points chosen to coincide with the interpolation points.

If the problem leaves us with some flexibility in the choice of interpolation points, then it is also possible to define the knots and deduce the optimal interpolation points. This is notably possible in the case of simulations where the initial condition is known analytically. In this case the same interpolation points must be used throughout, but the problem doesn't enforce their position.

Optimal interpolation points are usually considered to be the Greville abscissae, which are defined as:

$$x_i = \frac{\sum_{l=i+1}^{i+k} k_l}{d} \tag{2.46}$$

where k is the degree of the interpolation. This is because the spline which interpolates these points is bounded independent of the knot sequence for $d < 20$ (Jia 1986).

2.5.1. Boundary Conditions

As mentioned in section 2.3 a spline with no repeated knots in the domain has $n_c + d$ degrees of freedom. With n_p interpolation points $n_c + d - n_p$ additional conditions are

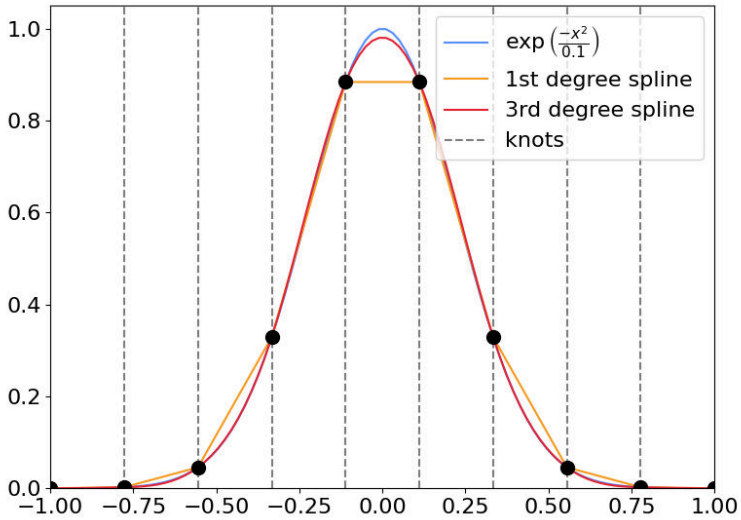


Figure 2.7.: Interpolation of the Gaussian function $\exp\left(\frac{-x^2}{0.1}\right)$ by a 1st degree and 3rd degree spline over 10 interpolation points

required in order to fully define the system. These additional conditions are usually provided by boundary conditions. In this section we will discuss several common choices.

2.5.1.1. Periodic Boundary Conditions

Periodic boundary conditions are commonly used to describe angles (poloidal or toroidal coordinates), or infinite spaces. They assume that the edges of the spline a and b both describe the same point. In order for the spline to remain $\mathcal{C}^{d-1}$ an additional continuity condition is required:

$$\partial_x^k S_{n_c-1,d}(z_{n_c}) = \partial_x^k S_{0,d}(z_0) \qquad \forall 0 \leq k < d \tag{2.47}$$

The b-splines are modified to include the additional continuity conditions at this point. The resulting b-splines can be seen in Figure 2.8. These additional d equations reduce the degrees of freedom of the system from $n_c + d$ to n_c.

Both the use of the break points as interpolation points, and the use of Greville abcissae lead to $n_p = n_c$ in this case, so no additional information is required to fully define the system.

2.5.1.2. Greville Boundary Conditions

For non-periodic domains, the choice of the boundary knots affects the number of Greville abcissae inside the domain. If the boundary knots are placed on the boundary as described by equation (2.15) then there are $n_p = n_c + d$ Greville abcissae.

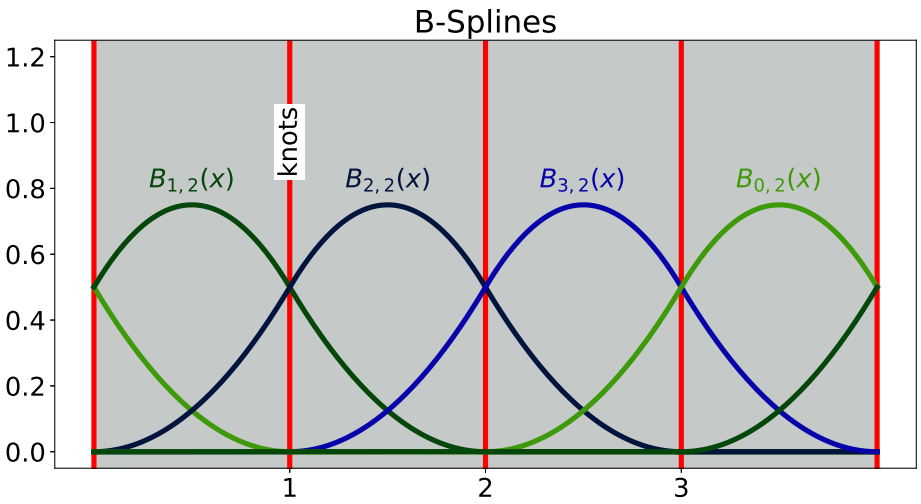


Figure 2.8.: B-splines of degree 2 for the domain $[0,4]$ with periodic boundary conditions

Here, as in the periodic case, no additional information is required to fully define the system.

2.5.1.3. Hermite Boundary Conditions

If however the interpolation points are placed at the break points, there are only n_c+1 interpolation points. Similarly, if the boundary knots are placed as described by equation (2.13) then there are only $n_c+(d \mod 2)$ interpolation points. An additional $d-1$ or $d-(d \mod 2)$ conditions are therefore required.

In the case of splines of even degree, placing interpolation points at the break points therefore leads to an unbalanced situation. An odd number of additional conditions must be imposed and it is not clear where this should be done. We will therefore avoid this case.

In the case of splines of odd degree $d-1 = d-(d \mod 2)$. There is therefore an even number of conditions to be imposed. There is also an even number of conditions to be imposed for even degree splines with Greville abcissae as interpolation points. In both cases $\frac{d-(d \mod 2)}{2}$ conditions are imposed at each boundary.

Hermite boundary conditions enforce the value and first m derivatives at the boundaries.

If there are already interpolation points located at the boundary, the values to be provided for Hermite boundary conditions are therefore:

$$\left.\begin{matrix}\partial_x^k f(a)\\ \partial_x^k f(b)\end{matrix}\right\} \qquad \forall\, 0 < k \leq \frac{d-(d \mod 2)}{2} \tag{2.48}$$

This is notably the case when break points are used as interpolation points. It is also

true when Greville abcissae are used with an odd degree if the knots are equidistant or the boundary knots are located at the boundary, as described by equation (2.15).

If however the interpolation points do not intersect with the boundaries, the values to be provided for Hermite boundary conditions are:

$$\left.\begin{matrix}\partial_x^k f(a)\\ \partial_x^k f(b)\end{matrix}\right\} \qquad \forall\, 0 \leq k < \frac{d-(d \mod 2)}{2} \tag{2.49}$$

This is the case for even degree splines with Greville abcissae used as interpolation points. It may also be true for splines with non-equidistant knots when the boundary knots are placed outside the domain.

2.5.1.4. Natural Boundary Conditions

Natural boundary conditions describe a situation very similar to Hermite boundary conditions in that the same number of conditions must be imposed in the same locations. In this case however it is not the first $\frac{d-(d \mod 2)}{2}$ derivatives which are provided, but the last $\frac{d-(d \mod 2)}{2}$ derivatives which are forced to zero. Thus the additional conditions for natural boundary conditions are:

$$\left.\begin{matrix}\partial_x^k f(a) = 0\\ \partial_x^k f(b) = 0\end{matrix}\right\} \qquad \forall\, \frac{d+(d \mod 2)}{2} \leq k < d \tag{2.50}$$

2.5.2. Coefficient Calculation

The coefficients c_i defined in equation (2.4) must be calculated to find the spline representation. These coefficients are found by solving a set of linear equations which can also be written in matrix form:

$$B\vec{c} = \vec{f} \tag{2.51}$$

The matrix B is known as the interpolation matrix. In the case of interpolating splines the equations to solve are the interpolation conditions (see equation (2.45)) and the boundary conditions.

For the simplest boundary conditions where no additional equations are required (periodic or Greville), the coefficients $c_{i,d}$ are the solution to the following matrix equation:

$$\begin{pmatrix} b_{0,d}(x_0) & b_{1,d}(x_0) & \dots & b_{n_b-1}(x_0) \\ b_{0,d}(x_1) & b_{1,d}(x_1) & \dots & b_{n_b-1}(x_1) \\ \vdots & & & \vdots \\ b_{0,d}(x_{n_p-1}) & b_{1,d}(x_{n_p-1}) & \dots & b_{n_b-1}(x_{n_p-1}) \end{pmatrix} \begin{pmatrix} c_{0,d} \\ c_{1,d} \\ \vdots \\ c_{n_b-1,d} \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{n_p-1}) \end{pmatrix}. \tag{2.52}$$

This matrix is invertible under the following condition (Hämmerlin and Hoffman

1991):

$$b_{j,d}(x_j) \neq 0 \qquad \forall 0 \leq j < n_b, \tag{2.53}$$

which is always satisfied for Greville abcissae or break points as interpolation points.

As mentioned in Section 2.3 the support of a b-spline $b_{i,d}(x)$ is $[k_i, k_{i+d+1}]$. This information can be used to reduce the memory footprint of the matrix as it can be represented as a banded matrix (or periodic banded in the case of periodic splines).

If the knots are used as interpolation points then the following is true:

$$b_{i,d}(x_j) \neq 0 \qquad \forall i+1 \leq j < i+d \tag{2.54}$$

The matrix is therefore banded with d diagonals.

If the Greville abcissae are used as interpolation points then the following is true:

$$k_{j+1} \leq x_j \leq k_{j+d} \tag{2.55}$$

$$b_{i,d}(x_j) \neq 0 \qquad \forall i \leq j < i+d+1 \tag{2.56}$$

For equidistant knots the Greville abcissae coincide with the knots for odd degree splines and are found at the centre of cells for even degree splines. As a result, the bounds can be reduced further:

$$b_{i,d}(x_j) \neq 0 \qquad \forall i \leq j < i+d+1-(d \mod 2) \tag{2.57}$$

For Greville boundary conditions with knots that are equidistant inside the domain, the first and last $d-1$ rows of the matrix respect equation (2.54) while all other rows respect equation (2.57). In order to minimise the necessary memory and reduce the number of FLOPS required to solve the system we leverage this information to write the system in block form:

$$B = \left(\begin{array}{c|c|c} \delta_1 & \lambda_1 & \delta_2 \\ \hline \gamma_1 & \mathrm{Q} & \gamma_2 \\ \hline \delta_3 & \lambda_2 & \delta_4 \end{array}\right) \tag{2.58}$$

Where Q is a banded matrix with d diagonals, and δ_1, δ_2, δ_3, δ_4, γ_1, γ_2, λ_1 and λ_2 are small sub-matrices. Depending on the chosen boundary conditions these sub-matrices may be known to only contain zeros.

For boundary conditions where additional equations are required to complete the matrix, this information will be contained in the sub-matrices. In this case the matrix remains invertible as long as the following condition is satisfied:

$$b_{j,d}(\xi_j) \neq 0 \qquad \forall 0 \leq j < n_b, \tag{2.59}$$

where ξ_j are the positions where the value or a derivative of the spline is fixed. For

example for Hermite boundary conditions on odd degree splines:

$$\xi_j = a \qquad \forall 0 \leq j < k \tag{2.60}$$
$$\xi_j = x_{j-k} \qquad \forall k \leq j < n_b - k \tag{2.61}$$
$$\xi_{n_b-j} = b \qquad \forall 0 \leq j < k \tag{2.62}$$

where $k = \frac{d-(d \mod 2)}{2}$ is the number of derivatives provided in this case. This condition is satisfied for all boundary conditions described in section 2.5.1.

For Hermite boundary conditions on odd degree splines the condition described by equation (2.48) can be expressed most simply as:

$$\left(\delta_1 \mid \lambda_1 \mid \delta_2 \right) = \begin{pmatrix} \partial_x b_{0,d}(a) & \partial_x b_{1,d}(a) & \dots & \partial_x b_{n_b-1}(a) \\ \partial_x^2 b_{0,d}(a) & \partial_x^2 b_{1,d}(a) & \dots & \partial_x^2 B_{n_b-1}(a) \\ \vdots & & & \vdots \\ \partial_x^k b_{0,d}(a) & \partial_x^k b_{1,d}(a) & \dots & \partial_x^k b_{n_b-1}(a) \end{pmatrix} \tag{2.63}$$

$$\left(\delta_3 \mid \lambda_2 \mid \delta_4 \right) = \begin{pmatrix} \partial_x b_{0,d}(b) & \partial_x b_{1,d}(b) & \dots & \partial_x b_{n_b-1}(b) \\ \partial_x^2 b_{0,d}(b) & \partial_x^2 b_{1,d}(b) & \dots & \partial_x^2 B_{n_b-1}(b) \\ \vdots & & & \vdots \\ \partial_x^k b_{0,d}(b) & \partial_x^k b_{1,d}(b) & \dots & \partial_x^k b_{n_b-1}(b) \end{pmatrix} \tag{2.64}$$

where $k = \frac{d-(d \mod 2)}{2}$.

For natural boundary conditions the condition described by equation (2.50) can be expressed most simply as:

$$\left(\delta_1 \mid \lambda_1 \mid \delta_2 \right) = \begin{pmatrix} \partial_x^l b_{0,d}(a) & \partial_x^l b_{1,d}(a) & \dots & \partial_x^l b_{n_b-1}(a) \\ \partial_x^{l+1} b_{0,d}(a) & \partial_x^{l+1} b_{1,d}(a) & \dots & \partial_x^{l+1} b_{n_b-1}(a) \\ \vdots & & & \vdots \\ \partial_x^d b_{0,d}(a) & \partial_x^d b_{1,d}(a) & \dots & \partial_x^d b_{n_b-1}(a) \end{pmatrix} \tag{2.65}$$

$$\left(\delta_3 \mid \lambda_2 \mid \delta_4 \right) = \begin{pmatrix} \partial_x^l b_{0,d}(b) & \partial_x^l b_{1,d}(b) & \dots & \partial_x b_{n_b-1}(b) \\ \partial_x^{l+1} b_{0,d}(b) & \partial_x^{l+1} b_{1,d}(b) & \dots & \partial_x^{l+1} b_{n_b-1}(b) \\ \vdots & & & \vdots \\ \partial_x^d b_{0,d}(b) & \partial_x^d b_{1,d}(b) & \dots & \partial_x^d b_{n_b-1}(b) \end{pmatrix} \tag{2.66}$$

where $l = \frac{d+(d \mod 2)}{2}$.

2.5.2.1. Solving the Spline Interpolation Matrix

The spline interpolation matrix as described by equation (2.51) has the following form:

$$B = \left(\begin{array}{c|c|c} \delta_1 & \lambda_1 & \delta_2 \\ \hline \gamma_1 & \mathrm{Q} & \gamma_2 \\ \hline \delta_3 & \lambda_2 & \delta_4 \end{array}\right) \tag{2.67}$$

where Q is a matrix with a narrow band, the matrices δ_1, λ_1, γ_1, γ_2, λ_2 and δ_4 describe the area of the matrix with a wider band, and δ_2 and δ_3 contain only zeros.

A simple exchange of indices allows us to re-arrange this into the following form:

$$B' = \left(\begin{array}{c|c|c} \mathrm{Q} & \gamma_1 & \gamma_2 \\ \hline \lambda_1 & \delta_1 & \delta_2 \\ \hline \lambda_2 & \delta_3 & \delta_4 \end{array}\right) \tag{2.68}$$

This matrix can be written more simply as follows:

$$B' = \left(\begin{array}{c|c} \mathrm{Q} & \gamma \\ \hline \lambda & \delta \end{array}\right) \tag{2.69}$$

A matrix equation $B'c' = f'$ where B' is of the form described above can be solved using Schur's complement.

The blockwise LU decomposition of the matrix is:

$$B' = LU \tag{2.70}$$

$$L = \left(\begin{array}{c|c} \mathrm{Q} & 0 \\ \hline \lambda & \delta' \end{array}\right) \tag{2.71}$$

$$U = \left(\begin{array}{c|c} \mathrm{I} & \beta \\ \hline 0 & I \end{array}\right) \tag{2.72}$$

where δ' is the Schur's complement of Q defined as $\delta' = \delta - \lambda\beta$, and β is the solution to the equation $Q\beta = \gamma$.

Finally, the solution to the equation $Bc = f$ can be calculated by splitting and reorganising the vectors c and f in a similar fashion:

$$c = \begin{pmatrix} e_1 \\ d \\ e_2 \end{pmatrix} \qquad c' = \begin{pmatrix} d \\ e_1 \\ e_2 \end{pmatrix} = \begin{pmatrix} d \\ e \end{pmatrix} \tag{2.73}$$

We can then solve the following equations:

1. $Lu = f'$:

$$\left(\begin{array}{c|c} \mathrm{Q} & 0 \\ \hline \lambda & \delta' \end{array}\right)\begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} g \\ h \end{pmatrix}$$

2. $Uc' = u$

$$\left(\begin{array}{c|c} \mathrm{I} & \beta \\ \hline 0 & I \end{array}\right)\begin{pmatrix} d \\ e \end{pmatrix} = \begin{pmatrix} v \\ w \end{pmatrix}$$

The steps are therefore as follows:

1. Solve $Qv = g$ for v
2. Solve $\delta' e = h - \lambda v$ for e
3. Calculate $d = v - \beta e$

2.5.2.2. Condition Number

It is important that the interpolation matrix is well conditioned so that the coefficients c_i can be calculated to an acceptable error range. In general k digits of accuracy of the solution of the matrix system are lost for a condition number $\kappa(A) = 10^k$(Cheney and Kincaid 2012). As these operations are conducted on a computer, any problem will always have an accuracy of at least machine precision ($\sim 10^{-16}$).

Spline interpolation problems tend to be well conditioned thanks to their minimal support and the partition of unity property. However the choice of boundary conditions can sometimes have a negative effect on the conditioning. This can be understood intuitively for Hermite or natural boundary conditions as the lines of the matrix pertaining to the boundary conditions are all non-zero in the same positions, making these lines less orthogonal than the rest of the matrix.

The conditioning of the matrices can also be improved somewhat through line normalisation. For natural boundary conditions, this can be done in the boundary region without worrying about the consequences for the right-hand side of the matrix equation as the right-hand side corresponding to these lines contains only zeros. Each line of the matrix is divided by its L-1 norm. For the central lines described by equation (2.52), the L-1 norm is equal to 1 thanks to the partition of unity normalisation as described by equation (2.7).

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

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

Proof. The ∞-norm of A is defined as:

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

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

$$\begin{aligned}
\kappa\left(M^{-1}A\right) &= \left\|M^{-1}A\right\|_\infty \left\|\left(M^{-1}A\right)^{-1}\right\|_\infty \\
&= \left\|A^{-1}M\right\|_\infty \\
&= \max_i \left\{ \sum_j \left| \left(a^{-1}\right)_{i,j} m_{j,j} \right| \right\} \\
&\leq \max_k \{m_{k,k}\} \max_i \left\{ \sum_j \left| a^{-1}_{i,j} \right| \right\} \\
&= \|A\|_\infty \left\|A^{-1}\right\|_\infty = \kappa(A)
\end{aligned} \tag{2.74}$$

□

As we will see in section 2.8, this normalisation is not always sufficient to obtain a well-conditioned matrix.

2.5.3. Coefficient Calculation for N-D Splines

The coefficients for N-D splines can be calculated similarly to in the 1D case, by solving a matrix equation as defined in equation (2.51). The main difference between the two cases is the size and sparsity pattern of the interpolation matrix.

As mentioned in section 2.4 the basis functions of N-D splines on domains without singular points are defined as products of 1-D basis splines. This information helps understand the sparsity pattern of the interpolation matrix. The matrix B is simply defined as the Kronecker product of the interpolation matrices B_i for 1D splines along each dimension i:

$$B = B_1 \otimes B_2 \otimes \cdots \otimes B_N \tag{2.75}$$

This configuration also allows some simplifications to be made when solving the matrix. Rather than solving for the coefficients in one calculation, it is possible to break the equation up into multiple problems equivalent to finding the coefficients of a 1D spline. This is possible due to the following identity:

$$B_1 \otimes B_2 \otimes \cdots \otimes B_N = (B_1 \otimes I_2 \otimes \cdots \otimes I_N) \ldots (I_1 \otimes \cdots \otimes I_{N-1} \otimes B_N) \tag{2.76}$$

For example in the case of a 2D spline with coefficients c_{ij} the coefficient calculation can be expressed as follows:

$$(B_1 \otimes I_2)(I_1 \otimes B_2)c = f \tag{2.77}$$

where c is the vector containing the coefficients c_{ij}, and f is the vector containing the values of the function at the interpolation points. By introducing an intermediate vector d we obtain the following 2 equations:

$$(B_1 \otimes I_2)d = f \tag{2.78}$$
$$(I_1 \otimes B_2)c = d \tag{2.79}$$

If we define d_i and f_i the sub-vectors of d, and f containing all the values corresponding to the i-th index in the first dimension we can write equation (2.78) as:

$$B_1 d_i = f_i \qquad \forall 0 \leq i < N_1 \tag{2.80}$$

Similarly, if we define c_j and d_j the sub-vectors of c, and d containing all the values corresponding to the j-th index in the second dimension we can write equation (2.79) as:

$$B_2 c_j = d_j \qquad \forall 0 \leq j < N_2 \tag{2.81}$$

In the case of $\mathcal{C}^k$ N-D splines with singular points, the Kronecker structure is not preserved over the first $\frac{(k+1)(k+2)}{2}$ rows and columns of the matrix, so this trick cannot be used and the matrix equation must be solved as written.

2.6. Spline Evaluation

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

$$S_d(x) = \sum_i c_i b_{i,d}(x) \tag{2.82}$$

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

$$S_d(x) = \sum_{i=i^*}^{i^*+d} c_i b_{i,d}(x) \tag{2.83}$$

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

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

$$i^* = \left\lfloor \frac{x-a}{h} \right\rfloor \tag{2.84}$$

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

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

i^* can therefore no longer be found trivially; instead a search algorithm is used. In the codes referenced in this thesis the binary search algorithm will be used. The average and worst cases for this algorithm are $\mathcal{O}(\log n)$ operations (Knuth 1998).

The remaining complexity in the evaluation of the spline comes from the calculation of the $d+1$ values $b_i^d(x)$ for $i^* \le i \le i^* + d$. The efficiency of this calculation depends on the constraints that are placed on the system.

In sections 2.6.1 - 2.6.3 I will describe three different constraints and their requirements in FLOPs. The performance of these methods will be examined in the context of a full simulation in chapter 3.

2.6.1. Cubic Uniform B-Splines

The most efficient calculation is available in the most restrictive case, where the algorithm is designed for a chosen degree and the knots are equidistant. The evaluation of the b-splines for cubic splines defined on equidistant knots is defined as follows:

$$o = (x - x_{i^\star})\, i\_dx \tag{2.85}$$
$$b = 1 - o \tag{2.86}$$
$$b_{i^\star-3,3}(x) = b^3 \tag{2.87}$$
$$b_{i^\star-2,3}(x) = 1 + 3(1 - b^2(2 - b)) \tag{2.88}$$
$$b_{i^\star-1,3}(x) = 1 + 3(1 - o^2(2 - o)) \tag{2.89}$$
$$b_{i^\star,3}(x) = o^3 \tag{2.90}$$

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

This calculation requires 19 FLOPs to calculate the necessary values $\{b_{i,d}(x)\}$.

2.6.2. Uniform B-Splines of Arbitrary Order

Although the use of cubic splines on uniform knots is highly optimised, when testing splines of different degrees it is simpler to make the degree a parameter of the algorithm.

The following algorithm calculates the value of the non-zero b-splines at a given evaluation point for a given degree:

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

Input: x : Evaluation point
Input: d : Spline degree
Input: $i^\star$: The index of the last knot before the evaluation point
Input: $k_{i^\star}$: The position of the last knot before the evaluation point
Input: Δx : The step between consecutive knots

```
v_1^0 = 1
o = (k_{i*} - x) / Δx
for j = 1, ..., d do
    s_0 = 0
    for r = 1, ..., j - 1 do
        v_r^j = s_{r-1} + (o + r) v_r^{j-1} / j
        s_r = (j - (o + r)) v_r^{j-1} / j
    end for
    v_r^j = s_k
end for
for j = 1, ..., d do
    b_{i*+j-1,d}(x) = v_k^d
end for
```

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

$$FLOPs = 1 + DIV + \sum_{j=1}^{d}\sum_{t=1}^{j} 5 + DIV = 5 + \sum_{j=1}^{d} 9j = 5 + 9\frac{d(d+1)}{2} \tag{2.91}$$

where the terms $\frac{v_r^{j-1}}{j}$ and $o+r$ are calculated to avoid repetition. A division is assumed to be equal to 4 FLOPs.

This calculation implies that 59 FLOPs are required for cubic splines.

The index and position of the last knot before the evaluation point, $i^\star$ and $k_{i^\star}$, can be calculated from the evaluation point x, and the step between consecutive knots:

$$i^\star = \lfloor \frac{x-a}{\Delta x} \rfloor, \tag{2.92}$$

$$k_{i^\star} = \Delta x i^\star. \tag{2.93}$$

However this calculation is not described in Algorithm 1 as it would not be fair to compare FLOPs calculated with these calculations to FLOPs calculated without the equivalent operation (the integer binary search) in the non-uniform case described by Algorithm 2.

2.6.3. Non-Uniform B-Splines of Arbitrary Order

Depending on the problem considered, we cannot always rely on having equidistant knots. The following algorithm describes the most general case where the degree is a parameter and the knots are non-equidistant. This algorithm was proposed in Algorithm A2.2 in Piegl and Tiller 1996. Algorithm 1 is a slightly optimised version of this algorithm which was obtained by simplifying expressions using the fact that the knots are equidistant.

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

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

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

$$FLOPs = \sum_{j=1}^{d} 2 + \sum_{t=1}^{j} 4 + DIV = 2d + 8\sum_{j=1}^{d} j = 2d + 8\frac{d(d+1)}{2} = 4d^2 + 6d \tag{2.94}$$

where the term $\frac{v_t^{j-1}}{r_t - l_{j-t}}$ is precalculated to avoid repetition. As previously, a division is assumed to be equivalent to 4 FLOPs.

This calculation implies that 54 FLOPs are required for cubic splines.

It is interesting to note that the non-equidistant case uses fewer FLOPs than the equidistant case, however it requires the storage of an additional $2d$ variables and the aforementioned binary search.

2.7. Spline Derivatives

As a b-spline is composed of polynomials which can be differentiated easily, it is equally simple to find the derivative of a spline. Taking the definition of a b-spline as written in equation (2.14) we see that:

$$\partial_x b_{i,1}(x) = \begin{cases} \frac{1}{k_{i+1}-k_i}\delta_{k_{i+1}\neq k_i} & \text{if } k_i \leq x < k_{i+1} \\ \frac{-1}{k_{i+2}-k_{i+1}}\delta_{k_{i+2}\neq k_{i+1}} & \text{if } k_{i+1} \leq x < k_{i+2} \\ 0 & \text{otherwise} \end{cases} \tag{2.95}$$

$$\begin{aligned} \partial_x b_{i,d}(x) =& \delta_{k_{i+d}\neq k_i}\left(\frac{1}{k_{i+d}-k_i}b_{i,d-1}(x) + \frac{x-k_i}{k_{i+d}-k_i}\partial_x b_{i,d-1}(x)\right) \\ &+ \delta_{k_{i+d+1}\neq k_{i+1}}\left(\frac{-1}{k_{i+d+1}-k_{i+1}}b_{i+1,d-1}(x) + \frac{k_{i+d+1}-x}{k_{i+d+1}-k_{i+1}}\partial_x b_{i+1,d-1}(x)\right) \end{aligned} \tag{2.96}$$

By induction it can be proven that this is equivalent to the following definition:

$$\frac{db_{i,d}(x)}{dx} = d\left(\delta_{k_{i+d}\neq k_i}\frac{b_{i,d-1}(x)}{k_{i+d}-k_i} - \delta_{k_{i+d+1}\neq k_{i+1}}\frac{b_{i+1,d-1}(x)}{k_{i+d+1}-k_{i+1}}\right) \tag{2.97}$$

Proof. In the following proof the Dirac functions are removed to simplify the expressions.

d=1

$$\frac{db_{i,1}(x)}{dx} = \left(\frac{b_{i,0}(x)}{k_{i+d}-k_i} - \frac{b_{i+1,0}(x)}{k_{i+d+1}-k_{i+1}}\right) \tag{2.98}$$

$$= \begin{cases} \frac{1}{k_{i+1}-k_i} & \text{if } k_i \leq x \leq k_{i+1} \\ \frac{-1}{k_{i+2}-k_{i+1}} & \text{if } k_{i+1} \leq x \leq k_{i+2} \\ 0 & \text{otherwise} \end{cases} \tag{2.99}$$

d+1

$$\begin{aligned} \partial_x b_{i,d+1}(x) =& \frac{b_{i,d}(x)}{k_{i+d+1}-k_i} + \frac{x-k_i}{k_{i+d+1}-k_i}\partial_x b_{i,d}(x) \\ &- \frac{b_{i+1,d}(x)}{k_{i+d+2}-k_{i+1}} + \frac{k_{i+d+2}-x}{k_{i+d+2}-k_{i+1}}\partial_x b_{i+1,d}(x) \\ =& \frac{b_{i,d}(x)}{k_{i+d+1}-k_i} + \frac{x-k_i}{k_{i+d+1}-k_i}d\left(\frac{b_{i,d-1}(x)}{k_{i+d}-k_i} - \frac{b_{i+1,d-1}(x)}{k_{i+d+1}-k_{i+1}}\right) \\ &- \frac{b_{i+1,d}(x)}{k_{i+d+2}-k_{i+1}} + \frac{k_{i+d+2}-x}{k_{i+d+2}-k_{i+1}}d\left(\frac{b_{i+1,d-1}(x)}{k_{i+1+d}-k_{i+1}} - \frac{b_{i+2,d-1}(x)}{k_{i+d+2}-k_{i+2}}\right) \end{aligned}$$

$$
\begin{aligned}
&= \frac{b_{i,d}(x)}{k_{i+d+1}-k_i} - \frac{b_{i+1,d}(x)}{k_{i+d+2}-k_{i+1}} \\
&\quad + d\left[\frac{(x-k_i)}{k_{i+d}-k_i}\frac{b_{i,d-1}(x)}{k_{i+d+1}-k_i} - \frac{k_{i+d+2}-x}{k_{i+d+2}-k_{i+2}}\frac{b_{i+2,d-1}(x)}{k_{i+d+2}-k_{i+1}}\right] \\
&\quad + d\frac{b_{i+1,d-1}(x)}{k_{i+d+1}-k_{i+1}}\left[\frac{(k_{i+d+2}-x)(k_{i+d+1}-k_i)-(x-k_i)(k_{i+d+2}-k_{i+1})}{(k_{i+d+2}-k_{i+1})(k_{i+d+1}-k_i)}\right] \\
&= \frac{b_{i,d}(x)}{k_{i+d+1}-k_i} - \frac{b_{i+1,d}(x)}{k_{i+d+2}-k_{i+1}} \\
&\quad + d\left[\frac{(x-k_i)}{k_{i+d}-k_i}\frac{b_{i,d-1}(x)}{k_{i+d+1}-k_i} - \frac{k_{i+d+2}-x}{k_{i+d+2}-k_{i+2}}\frac{b_{i+2,d-1}(x)}{k_{i+d+2}-k_{i+1}}\right] \\
&\quad + d\frac{b_{i+1,d-1}(x)}{k_{i+d+1}-k_{i+1}}\left[\frac{(k_{i+d+2}-k_{i+1})(k_{i+d+1}-x)-(k_{i+d+1}-k_i)(k_{i+d+2}-x)}{(k_{i+d+2}-k_{i+1})(k_{i+d+1}-k_i)}\right] \\
&= \frac{b_{i,d}(x)}{k_{i+d+1}-k_i} - \frac{b_{i+1,d}(x)}{k_{i+d+2}-k_{i+1}} \\
&\quad + \frac{d}{k_{i+d+1}-k_i}\left[\frac{x-k_i}{k_{i+d}-k_i}b_{i,d-1}(x) + \frac{k_{i+d+1}-x}{k_{i+d+1}-k_{i+1}}b_{i+1,d-1}(x)\right] \\
&\quad - \frac{d}{k_{i+d+2}-k_{i+1}}\left[\frac{x-k_{i+1}}{k_{i+1+d}-k_{i+1}}b_{i+1,d-1}(x) + \frac{k_{i+d+2}-x}{k_{i+d+2}-k_{i+2}}b_{i+2,d-1}(x)\right] \\
&= \frac{b_{i,d}(x)}{k_{i+d+1}-k_i} - \frac{b_{i+1,d}(x)}{k_{i+d+2}-k_{i+1}} + \frac{d}{k_{i+d+1}-k_i}b_{i,d} - \frac{d}{k_{i+d+2}-k_{i+1}}b_{i+1,d} \\
&= (d+1)\left(\frac{b_{i,d}(x)}{k_{i+d+1}-k_i} - \frac{b_{i+1,d}(x)}{k_{i+d+2}-k_{i+1}}\right)
\end{aligned}
\tag{2.100}
$$

□

Equation (2.97) can easily be evaluated using the algorithms described in Section 2.6.

2.8. Spline Integration

The integral of a spline, defined by equation (2.4), is expressed simply as:

$$\int_a^b S_d(x)\,dx = \sum_{i=0}^{n_b-1} c_i \int_a^b b_{i,d}(x)\,dx \tag{2.101}$$

where c_i are the coefficients of the spline approximation and $b_{i,d}(x)$ are the b-splines.

The integral of a b-spline $b_{i,d}(x)$ over its support can be expressed using equation (2.9) which relates b-splines respecting partition of unity ($b_{i,d}(x)$) to b-splines normalised to have unit integral($Q_{i,d}$):

$$\int_{k_i}^{k_{i+d+1}} b_{i,d}(x)\,dx = \frac{k_{i+d+1}-k_i}{d+1}\int_{k_i}^{k_{i+d+1}} Q_{i,d}(x)\,dx = \frac{k_{i+d+1}-k_i}{d+1} \tag{2.102}$$

The case is slightly more complicated for b-splines in the boundary region with boundary knots outside the domain, or when integrating over a part of the domain. In this case we cannot necessarily integrate the b-spline over its entire support. To handle these cases we use the following equation (Boor, Lyche, and Schumaker 1976):

$$\int_u^v b_{i,d}(x)dx = \frac{k_{i+d+1} - k_i}{d+1}\left(\sum_{j\geq i} b_{j+1,d+1}(v) - \sum_{j\geq i} b_{j+1,d+1}(u)\right) \tag{2.103}$$

The integration of the b-spline only makes sense for values within its domain. I.e. for u (and v) such that $b_{i,d}(u) \neq 0$, and $k_i \leq u \leq k_{i+d+1}$. In this case we can rewrite equation (2.103) as follows:

$$\int_u^v b_{i,d}(x)dx = \frac{k_{i+d+1} - k_i}{d+1}\left(\sum_{j=i}^{i+d}\left[b_{j+1,d+1}(v) - b_{j+1,d+1}(u)\right]\right) \tag{2.104}$$

2.8.1. Spline Quadrature

The integral of a function $f(x)$ on a domain $[a,b]$ can be approximated by calculating the integral of its interpolating spline on that domain:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n_b-1} c_i \int_a^b b_{i,d}(x)dx \tag{2.105}$$

where n_b is the number of basis functions.

Let us define the constants $I(b_i) = \int_a^b b_{i,k}(x)dx$. As the spline is an interpolating spline, the coefficients c_i are obtained by solving the matrix equation (2.51). The equation can therefore be written as:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n_b-1}\sum_{j=0}^{n_b-1}\left(B^{-1}\right)_{ij} f_j I(b_i) \tag{2.106}$$

$$= \sum_{i=0}^{n_b-1}\sum_{j=0}^{n_b-1}\left(B^{-1}\right)^T_{ji} I(b_i) f_j \tag{2.107}$$

By calculating the coefficients $q_i = \sum_j \left(B^{-1}\right)^T_{ji} b_i$ in advance and choosing boundary conditions which do not require derivatives to be provided, this expression can be reduced to a quadrature formulation:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n_b-n_{bc}} q_{i+n_{bc}} f_i \tag{2.108}$$

where n_{bc} is the number of derivatives fixed at each edge. For example, in the case of natural boundary conditions on splines of odd degree: $n_{bc} = \frac{d-1}{2}$.

The quadrature coefficients can be calculated simply by solving the following matrix

equation:

$$B^T \vec{q} = \vec{I}_b \tag{2.109}$$

where $\vec{I}_b$ is the vector containing the constants $I(b_i)$.

2.8.2. "Best" Quadrature

Schoenberg has shown that the "best" quadrature (by which we mean the quadrature which minimises the error for a given degree d and set of points $\{x_i\}$) is one derived by integrating a spline approximation of the function interpolated at the break points, with natural boundary conditions such that the spline is of degree d at the edge of the domain (Schoenberg 1964).

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

The standard definition of the interpolation matrix B is described in section 2.5.2, with the boundary conditions being described by equations (2.65) and (2.66). As the derivatives are all set to zero the conditioning of the matrix can be improved as described in section 2.5.2.2. However specifying high derivatives in this way tends to be a poorly conditioned problem. Although the condition number is improved it still remains quite high. I will refer to this method as the "intuitive" method.

An alternative method for determining the quadrature coefficients was proposed by Secrest 1965. However, this method is also poorly conditioned (see table 2.1). The conditioning of interpolation matrices tends to increase with the degree of the spline being interpolated. As a result this is a critical problem for this method where large degrees are required.

I propose a new formulation which is significantly less susceptible to conditioning problems.

2.8.2.1. Proposed Implementation

My method is based on two observations. Firstly, that the $(d+1)$-th derivative of a spline of degree $2d+1$ is a spline of degree d. I will note this spline as $T_d(x)$. In order to respect the boundary conditions described by equation (2.50) this spline must respect the following equations:

$$\partial_x^i T_d(k_0) = \partial_x^i T_d(k_{N_c}) = 0 \quad \forall 0 \leq i < d \tag{2.110}$$

Secondly, I observe that in order to respect equation (2.110), the first d coefficients of the spline $T_d(x)$ must be equal to 0.

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

and Hoffman 1991):

$$\partial_x b_{i,p}(x) = p\left(\frac{b_{i,p-1}(x)}{k_{i+p}-k_i} - \frac{b_{i+1,p-1}(x)}{k_{i+1+p}-k_{i+1}}\right) \tag{2.111}$$

Note that b-splines of lower order require fewer knots to describe the system. Thus if the b-splines are defined using equation (2.11) with the knots which describe a spline of degree $r = 2d+1$, the b-splines $b_{i,r-j}$ are outside the domain $[a,b]$ for all $i < j$ and $N-i < N-j$.

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

$$\partial_x S_p(x) = \sum_{i=r-p+1}^{N_c+r-1} c_i^{[r-p+1]} b_{i,p-1}(x) \tag{2.112}$$

$$= \sum_{i=r-p}^{N_c+r-1} c_i^{[r-p]} \partial_x b_{i,p}(x) \tag{2.113}$$

$$= \sum_{i=r-p}^{N_c+r-1} c_i^{[r-p]} p\left(\frac{b_{i,p-1}(x)}{k_{i+p}-k_i} - \frac{b_{i+1,p-1}(x)}{k_{i+1+p}-k_{i+1}}\right) \tag{2.114}$$

$$= c_{r-p}^{[r-p]} p\frac{b_{r-p,p-1}(x)}{k_r - k_{r-p}} - c_{N_c+r-1}^{[r-p]} p\frac{b_{N_c+r,p-1}(x)}{k_{N_c+r+p}-k_{N_c+r}} + \sum_{i=r-p+1}^{N_c+r-1} p\frac{c_i^{[r-p]} - c_{i-1}^{[r-p]}}{k_{i+p}-k_i} b_i^{p-1}(x) \tag{2.115}$$

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

$$c_i^{[j]} = (r-j+1)\frac{c_i^{[j-1]} - c_{i-1}^{[j-1]}}{k_{i+r-j+1}-k_i} \tag{2.116}$$

Equation (2.110) implies:

$$c_{d+1+i}^{[d+1]} = 0 \quad \forall 0 \le i < d \tag{2.117}$$

$$c_{N_c+2d+1-i}^{[d+1]} = 0 \quad \forall 0 \le i < d \tag{2.118}$$

A combination of equations (2.116), (2.117), and (2.118) provides the equations required to fill the first and last d rows of the interpolation matrix.

Due to the recursive nature of equation (2.116), I will refer to this method as the "recursive" method.

The algorithm which constructs the necessary matrix is based on the fact that equation (2.116) can be expressed as a matrix equation:

$$c_i^{[j]} = \begin{bmatrix} -\frac{(r-j+1)}{k_{i+r-j+1}-k_i} & \frac{(r-j+1)}{k_{i+r-j+1}-k_i} \end{bmatrix} \begin{pmatrix} c_{i-1}^{[j-1]} \\ c_i^{[j-1]} \end{pmatrix} \tag{2.119}$$

$$= \frac{(r-j+1)}{k_{i+r-j+1}-k_i} \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} -\frac{(r-j+2)}{k_{i+r-j+1}-k_{i-1}} & \frac{(r-j+2)}{k_{i+r-j+1}-k_{i-1}} \\ -\frac{(r-j+2)}{k_{i+r-j+2}-k_i} & \frac{(r-j+2)}{k_{i+r-j+2}-k_i} \end{bmatrix} \begin{pmatrix} c_{i-2}^{[j-2]} \\ c_{i-1}^{[j-2]} \\ c_i^{[j-2]} \end{pmatrix} \tag{2.120}$$

The algorithm therefore loops over j calculating the line of the matrix representing $c_i^{[d+1]}\left(c_{i-j}^{[d+1-j]},\ldots,c_i^{[d+1-j]}\right)$ until $j = d+1$.
The algorithm is as follows:

Algorithm 3 My algorithm for calculating the line matrix describing natural boundary conditions

---

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

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

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

This block matrix has the following sparsity pattern (example given for a spline of

degree 7 at the bound x_0):

$$\begin{pmatrix} * & * & * & * & * & & & \dots \\ & * & * & * & * & * & & \dots \\ & & * & * & * & * & * & \dots \end{pmatrix}$$

Whereas the intuitive method has the following sparsity pattern:

$$\begin{pmatrix} * & * & * & * & * & * & * & \dots \\ * & * & * & * & * & * & * & \dots \\ * & * & * & * & * & * & * & \dots \\ * & * & * & * & * & * & * & \dots \end{pmatrix} \tag{2.121}$$

The increased orthogonality of the equations leads to a smaller condition number as we will see in section 2.8.2.2.

2.8.2.2. Precision Comparison

We therefore have 3 methods to compare. The first is the method proposed by Secrest (Secrest 1965), the second is the intuitive method described by equations (2.65) and (2.66), and the third is the recursive method described by equations (2.117), and (2.118). As explained in section 2.5.2.2, the condition number of the matrix required to calculate the quadrature coefficients is related to the final precision obtained after solving the equation. Supposing we desire a final precision of 10^{-8} any matrix with a condition number $\kappa(A) > 10^8$ would be unusable.

Table 2.1 shows the conditioning of the three different methods. We can see that Secrest's method is very poorly conditioned, especially for large problems. For the requested precision we are limited to low quadrature degrees and few points. In contrast the two b-spline based methods perform much better when the number of points is increased. These methods remove the dependency of the condition number on the problem size which allows much larger problems to be tackled effectively. Although the intuitive method performs significantly better than Secrest's method for a large number of points, the condition number still increases rapidly with the degree. As a result for the requested precision we would still be limited to 5th order quadrature. In contrast my proposed recursive method does not suffer from this problem. While the condition number also increases with the degree it does so gradually. As a result, for the requested precision, quadrature coefficients can be calculated up to 20th order. The small condition number for low degrees also means that we can be sure that the calculation of the quadrature coefficients will not introduce significant errors into the final quadrature calculation.

2.8.2.3. Newton-Cotes Quadrature

In order to evaluate the spline quadrature, a comparison point is needed. Three non-uniform Newton-Cotes schemes are used to provide this comparison: the well-

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

Table 2.1.: Table showing the conditioning of the matrix required to find the coefficients of a natural spline with equidistant knots on the domain $[-1,1]$ calculated using three different methods

known trapezoid rule, Simpson's rule, and Boole-Villarceau's rule. These quadrature methods are usually expressed on uniform points. Shklov 1960 explains how to express Simpson's rule on non-uniform points. The expression for Boole-Villarceau's rule cannot be found in the literature, so I derive it here.

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

$$\int f(x)dx \approx \frac{4h}{90} \sum_{i=0}^{N/4-1} \left(7f_{4i} + 32f_{4i+1} + 12f_{4i+2} + 32f_{4i+3} + 7f_{4i+4}\right) \tag{2.122}$$

where $h = h_i = x_{i+1} - x_i$, and $f(x_i) = f_i$.

The non-equidistant version has the following form:

$$\int f(x)dx \approx \sum_{i=0}^{N/4-1} \left(\alpha_i f_{4i} + \beta_i f_{4i+1} + \gamma_i f_{4i+2} + \zeta_i f_{4i+3} + \eta_i f_{4i+4}\right) \tag{2.123}$$

The coefficients α_i, β_i, γ_i, ζ_i and η_i are chosen to minimise the error.

The problem is simplified by examining a small section of the integral:

$$\begin{aligned}
&\frac{1}{h_1+h_2+h_3+h_4}\left[\int_{x-h_1-h_2}^{x+h_3+h_4} f(x)dx\right. \\
&\left.-\alpha f(x-h_1-h_2)-\beta f(x-h_2)-\gamma f(x)-\zeta f(x+h_3)-\eta f(x+h_3+h_4)\right] && (2.124)\\
=&\frac{1}{\sum_{i=1}^{4} h_i}\left[F(x+h_3+h_4)-F(x-h_1-h_2)\right. \\
&\left.-\alpha f(x-h_1-h_2)-\beta f(x-h_2)-\gamma f(x)-\zeta f(x+h_3)-\eta f(x+h_3+h_4)\right] && (2.125)\\
=&\frac{1}{\sum_{i=1}^{4} h_i}\left[F(x)+\sum_{i=0}^{\infty}\frac{(h_3+h_4)^{i+1}}{(i+1)!}f^{(i)}(x)\right. \\
&-F(x)+\sum_{i=0}^{\infty}(-1)^i\frac{(h_1+h_2)^{i+1}}{(i+1)!}f^{(i)}(x)-\alpha\left(\sum_{i=0}^{\infty}(-1)^i\frac{(h_1+h_2)^i}{i!}f^{(i)}(x)\right) \\
&-\beta\left(\sum_{i=0}^{\infty}(-1)^i\frac{h_2^i}{i!}f^{(i)}(x)\right)-\gamma f(x)-\zeta\left(\sum_{i=0}^{\infty}\frac{h_3^i}{i!}f^{(i)}(x)\right) \\
&\left.-\eta\left(\sum_{i=0}^{\infty}\frac{(h_3+h_4)^i}{i!}f^{(i)}(x)\right)\right] && (2.126)
\end{aligned}$$

A series of equations can now be defined to determine the coefficients which minimise the error:

$$h_3+h_4+h_1+h_2-\alpha-\beta-\gamma-\zeta-\eta=0 \quad (2.127)$$

$$\frac{(h_3+h_4)^2}{2}-\frac{(h_1+h_2)^2}{2}+\alpha(h_1+h_2)+\beta h_2-\zeta h_3-\eta(h_3+h_4)=0 \quad (2.128)$$

$$\frac{(h_3+h_4)^3}{3!}+\frac{(h_1+h_2)^3}{3!}-\alpha\frac{(h_1+h_2)^2}{2}-\beta\frac{h_2^2}{2}-\zeta\frac{h_3^2}{2}-\eta\frac{(h_3+h_4)^2}{2}=0 \quad (2.129)$$

$$\frac{(h_3+h_4)^4}{4!}-\frac{(h_1+h_2)^4}{4!}+\alpha\frac{(h_1+h_2)^3}{3}+\beta\frac{h_2^3}{3}-\zeta\frac{h_3^3}{3}-\eta\frac{(h_3+h_4)^3}{3}=0 \quad (2.130)$$

$$\frac{(h_3+h_4)^5}{5!}+\frac{(h_1+h_2)^5}{5!}-\alpha\frac{(h_1+h_2)^4}{4}-\beta\frac{h_2^4}{4}-\zeta\frac{h_3^4}{4}-\eta\frac{(h_3+h_4)^4}{4}=0 \quad (2.131)$$

This gives the following expressions:

$$\begin{aligned}
\alpha=&\frac{\sum_{i=1}^{4} h_i}{60h_1(h_1+h_2)(h_1+h_2+h_3)}\left(12h_1^3+21h_1^2h_2+6h_1^2h_3-9h_1^2h_4+6h_1h_2^2\right. \\
&+2h_1h_2h_3-8h_1h_2h_4-4h_1h_3^2+2h_1h_3h_4+6h_1h_4^2-3h_2^3-4h_2^2h_3 \\
&\left.+h_2^2h_4+h_2h_3^2+2h_2h_3h_4+h_2h_4^2+2h_3^3+h_3^2h_4-4h_3h_4^2-3h_4^3\right) && (2.132)
\end{aligned}$$

$$\beta = \frac{\left(\sum_{i=1}^{4} h_i\right)^3}{60 h_1 h_2 \left(h_2+h_3\right)\left(h_2+h_3+h_4\right)} \left(3h_1^2 + 6h_1h_2 + h_1h_3 - 4h_1h_4 + 3h_2^2 + h_2h_3 - 4h_2h_4 - 2h_3^2 + h_3h_4 + 3h_4^2\right) \tag{2.133}$$

$$\gamma = -\frac{\left(\sum_{i=1}^{4} h_i\right)^3}{60 h_2 h_3 \left(h_1+h_2\right)\left(h_3+h_4\right)} \left(3h_1^2 + h_1h_2 + h_1h_3 - 4h_1h_4 - 2h_2^2 - 4h_2h_3 + h_2h_4 - 2h_3^2 + h_3h_4 + 3h_4^2\right) \tag{2.134}$$

$$\zeta = \frac{\left(\sum_{i=1}^{4} h_i\right)^3}{60 h_3 h_4 \left(h_2+h_3\right)\left(h_1+h_2+h_3\right)} \left(3h_1^2 + h_1h_2 - 4h_1h_3 - 4h_1h_4 - 2h_2^2 + h_2h_3 + h_2h_4 + 3h_3^2 + 6h_3h_4 + 3h_4^2\right) \tag{2.135}$$

$$\eta = -\frac{\left(\sum_{i=1}^{4} h_i\right)}{60 h_4 \left(h_3+h_4\right)\left(h_2+h_3+h_4\right)} \left(3h_1^3 + 4h_1^2h_2 - h_1^2h_3 - 6h_1^2h_4 - h_1h_2^2 - 2h_1h_2h_3 - 2h_1h_2h_4 - h_1h_3^2 + 8h_1h_3h_4 + 9h_1h_4^2 - 2h_2^3 - h_2^2h_3 + 4h_2^2h_4 + 4h_2h_3^2 - 2h_2h_3h_4 - 6h_2h_4^2 + 3h_3^3 - 6h_3^2h_4 - 21h_3h_4^2 - 12h_4^3\right) \tag{2.136}$$

For equidistant points this simplifies to:

$$\alpha = \frac{14h}{45} = 4h\frac{7}{90} \tag{2.137}$$

$$\beta = \frac{64h}{45} = 4h\frac{32}{90} \tag{2.138}$$

$$\gamma = \frac{8h}{15} = 4h\frac{12}{90} \tag{2.139}$$

$$\zeta = \frac{64h}{45} = 4h\frac{32}{90} \tag{2.140}$$

$$\eta = \frac{14h}{45} = 4h\frac{7}{90} \tag{2.141}$$

The expression is $\mathcal{O}\left(h^6\right)$ for non-equidistant points and $\mathcal{O}\left(h^7\right)$ for equidistant points. The simplified case is equivalent to the expected result from equation 2.122.

2.8.2.4. Convergence

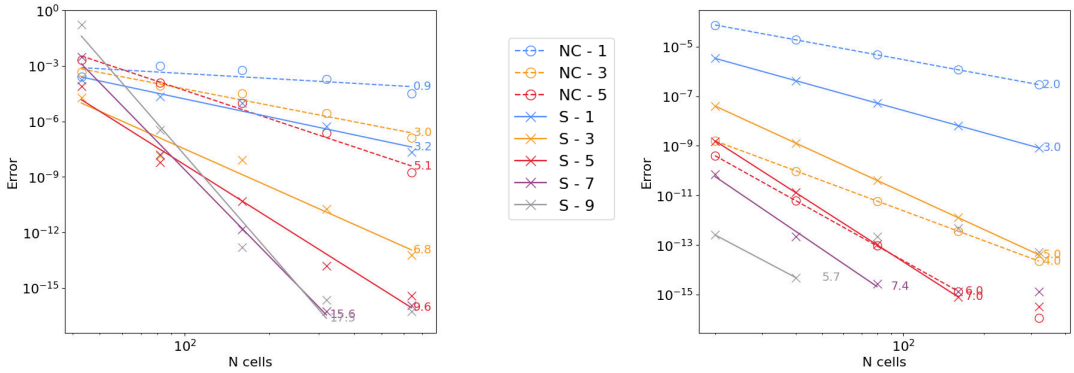
The convergence of the quadrature scheme proposed in Section 2.8.2.1 is investigated using equations for which the exact integral is known. The first equation is a Maxwellian test function used in Section 3.4.1 to describe a distribution function, the second is an integral suggested by Bailey, Jeyabalan, and Li 2011 for testing quadrature:

$$\int_{-30}^{30} \frac{1}{2\pi} \exp\left(-\frac{x^2}{2}\right) dx = \frac{1}{\sqrt{2\pi}} \operatorname{erf}\left(\frac{30}{\sqrt{2}}\right) \tag{2.142}$$

$$\int_0^1 \frac{\arctan\left(2+t^2\right)}{\left(1+t^2\right)\sqrt{2+t^2}} dx = 5\pi^2/96 \tag{2.143}$$

The results can be seen in Figure 2.9. We see that in the Maxwellian case examined in Figure 2.9a where the natural boundary conditions provide a good representation of the actual boundary conditions, the order of convergence of the spline scheme is closer to the order of the underlying spline. As a result the spline scheme performs significantly better than the Newton-Cotes schemes.

In contrast, in Figure 2.9b we see that the spline quadrature does not always perform better when integrating equation 2.143. However as it is much easier to implement higher-order non-uniform schemes with this method it is easy to use higher-order schemes to obtain improved errors.



(a) Equation (2.142). (b) Equation (2.143).

Figure 2.9.: The convergence of the error when solving Equations (2.142) and (2.143) with different degree Newton-Cotes (NC) or spline (S) quadrature schemes.

2.9. Conclusion

In this chapter, the various aspects of splines used in this work have been explained. This will form the basis for most of the methods described in the rest of this thesis. Interpolating splines are used in Chapters 3 and 4 to underpin the semi-Lagrangian method. Splines are used in the finite element method in Chapters 3, 5, and 6, in particular $\mathcal{C}^1$ polar splines are used in Chapters 5, and 6.

A new approach to Schoenberg's "best" quadrature was presented. The new method proved to massively improve the conditioning of the problem, improving the precision of the quadrature coefficients by several significant figures. For example, in the case of the fifth order scheme, four additional significant figures of precision are obtained. This quadrature is used in Chapter 3 in the context of plasma sheath simulations.

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

3.1. Introduction

This chapter reproduces the work of Bourne, Munschy, Virginie Grandgirard, et al. 2022. In it, non-uniform splines of varying degrees are used to run kinetic semi-Lagrangian simulations of the plasma sheath.

The plasma sheath is the part of a hot plasma adjacent to a comparatively cold wall. In this area there are steep gradients (Guilhem Dif-Pradalier, Philippe Ghendrih, Yanick Sarazin, et al. 2022), for example in temperature, and density, compared to the rest of the plasma (Coulette and Giovanni Manfredi 2014). These constraints make this area particularly difficult to simulate. The behaviour in this area is inherently kinetic since particle velocity plays a strong role in establishing the plasma current. Furthermore, electrons and ions, having different average velocities, exhibit distinct behaviour. It is this difference in behaviour which leads to the formation of the sheath; therefore each species must be modelled individually. This precludes the use of simpler models such as fluid models and forces the use of more complicated kinetic models.

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

When using equidistant sampling points, such a small resolution comes with a very large cost both in computation and memory consumption. This is especially the case

for large multi-dimensional simulations such as the GYSELA code (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016) which simulates the entirety of a tokamak in five dimensions. Such simulations already operate at the bounds of what is possible on today's supercomputers. It is therefore not feasible for these simulations to drastically increase the number of points used in the simulation. Non-equidistant sampling points allow the steep gradient to be resolved without unduly increasing the memory required for the simulation. This can cause significant memory gains in a five dimensional simulation as the removal of just one point in a given dimension, reduces the data required by $n_2 n_3 n_4 n_5$ points, where n_2, n_3, n_4, and n_5 are the number of points in the other four dimensions.

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

In this chapter we will investigate the changes necessary to move from a semi-Lagrangian simulation on equidistant points to one with non-equidistant points, with a view to extending the GYSELA code (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016) to more accurately simulate the edge domain of a tokamak plasma. Given the complexity of the GYSELA code, a reduced model based on the same algorithms is used to investigate this problem and determine whether the memory-calculation trade-off is acceptable. This reduced model will consider only the sheath, which is a complicated problem in its own right having been investigated in multiple other papers (Coulette and Giovanni Manfredi 2014; G. Manfredi and Valsaque 2004; Manfredi, Hirstoaga, and Devaux 2010; Badsi, M., Mehrenberger, M., and Navoret, L. 2018). The handling of the wall in this area as well as steep gradients in the sheath region itself mean that this simulation is highly pertinent for the testing of non-equidistant points, in addition to actually describing the region that will be added to GYSELA.

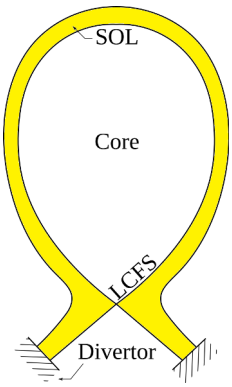
Previous works simulating the sheath with non-uniform points have used finite volumes methods (Coulette and Giovanni Manfredi 2014; G. Manfredi and Valsaque 2004); however this gives rise to time and spatial step restrictions. The GYSELA code (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016) takes a different approach using a semi-Lagrangian method to reduce these restrictions. This is crucial in such a large code in order to facilitate running simulations over large time scales. Previous works using a semi-Lagrangian method for sheath simulations include the work of Badsi, M., Mehrenberger, M., and Navoret, L. 2018. They use Lagrange interpolation on uniform points to reconstruct the function, whereas in this chapter, as in the GYSELA code (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016), a spline interpolation will be used. Manfredi, Hirstoaga, and Devaux 2010 also studied the sheath on uniform points, using a semi-Lagrangian method, additionally using a slope corrector to ensure the positivity of the distribution function. Semi-Lagrangian advection on cubic splines with non-equidistant points in the velocity dimension has previously been investigated by Afeyan et al. in a different context: the study of Kinetic Electrostatic Electron

Nonlinear (KEEN) waves (Afeyan, Casas, Nicolas Crouseilles, et al. 2014).

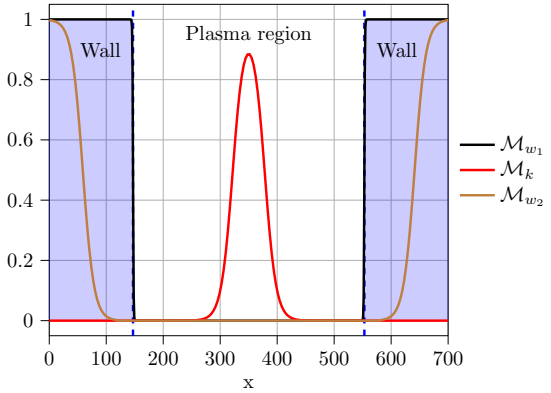
This work distinguishes itself from previous work by the use of non-equidistant points in all dimensions, Greville abcissae to avoid explicit boundary conditions, and the use of the "best" quadrature in the sense of Schoenberg as described in chapter 2. This quadrature is rarely used as it is difficult to calculate the quadrature coefficients to a sufficient precision. In section 2.8 we presented a new method to calculate these coefficients, which solves this problem.

This chapter presents work which is organised as follows. Section 3.2 presents the physical system being modelled. Section 3.3 presents the different schemes used to resolve the system. Section 3.4 discusses the convergence of the different methods for non-equidistant points. Section 3.5 presents the non-equidistant simulations and discusses the gains that were achieved as compared to equidistant points. Finally section 3.6 gives our conclusions.

3.2. Physical System



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



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

Figure 3.1.: The 1D spatial simulation domain.

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

The sheath model used for the simulations in this chapter describes a highly simplified situation. The domain is considered to be one-dimensional. This dimension describes a line which follows a magnetic field line, beginning and ending in a wall. This situation can be seen in Figure 3.1 (see Figure 3.1a for a 2D diagram of a tokamak plasma cross-section, and Figure 3.1b for the 1D simulation domain). The simulation consists of three regions, two wall regions and the plasma region.

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

As a result of this behaviour the electric potential has a very steep gradient close to the wall (Guilhem Dif-Pradalier, Philippe Ghendrih, Yanick Sarazin, et al. 2022). This can be understood as a standing shock wave localised at the wall. In order to take into account the behaviour due to this gradient it is important to resolve the system with a small spatial resolution in this area. This prevents the wall being seen as a discontinuity in the distribution function. Discontinuities can propagate through a simulation causing non-physical behaviour such as oscillations.

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

3.2.1. Vlasov-Poisson

The system is known as the Vlasov-Poisson system as it is composed of a Vlasov equation and a Poisson equation. The Vlasov equation (3.1) is an advection equation describing the evolution of the distribution function $f_s(t,x,v)$ for electrons ($s=e$), and ions ($s=i$). The Poisson equation (3.2) describes the quasi-neutrality condition:

$$\partial_t f_s(t,x,v) + v\partial_x f_s(t,x,v) - \frac{q_s}{m_s}\partial_x\phi(t,x)\partial_v f_s(t,x,v) = S_s(t,x,v) + C_{ss}(t,x,v) \quad (3.1)$$

$$\partial_x^2\phi(t,x) = -\frac{\rho_q(t,x)}{\varepsilon_0} \quad (3.2)$$

Where x is the spatial coordinate, v is the velocity coordinate, t is the temporal coordinate, q_s and m_s are respectively the charge and mass of the species s, $S_s(t,x,v)$ is a source term, $C_{ss}(t,x,v)$ is a intra-species collision operator, $\phi(t,x)$ is the electric potential, ρ_q is the charge density, and ε_0 is the permittivity of free space.

The charge density ρ_q is defined from the particle density n_s as follows:

$$\rho_q(t,x) = \sum_s q_s n_s(t,x) \tag{3.3}$$

$$n_s(t,x) = \int f_s(t,x,v)\,dv \tag{3.4}$$

The collision operator $C_{ss}(t,x,v)$ resembles the operator used in GYSELA (G. Dif-Pradalier, Diamond, V. Grandgirard, et al. 2011), and is defined as follows:

$$C_{ss}(t,x,v) = \partial_v \left[D_v(t,x,y(x,v))\partial_v f_s(t,x,v) + f_s(t,x,v) D_v(t,x,y(x,v)) m_s \frac{v - V_M(t,x)}{T_M(t,x)} \right] \tag{3.5}$$

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

$$D_v(t,x,y(x,v)) = D_0(t,x) \left[\frac{\psi(y)}{y} - \frac{\psi(y)}{2y^3} + \frac{\psi'(y)}{y^2} \right] \tag{3.6}$$

$$D_0(t,x) = \frac{3\sqrt{2\pi} T_s(t,x)}{4 m_s} \nu_{ss} \tag{3.7}$$

$$T_s(t,x) = \int \frac{m_s (v - V_s(t,x))^2}{n_s(t,x)} f_s(t,x,v)\,dv \tag{3.8}$$

$$V_s(t,x) = \int \frac{v}{n_s(t,x)} f_s(t,x,v)\,dv \tag{3.9}$$

$$\psi(y) = \frac{2}{\sqrt{\pi}} \int_0^y \exp\left(-z^2\right) dz \tag{3.10}$$

$$y(x,v) = \frac{|v|\sqrt{m_s}}{\sqrt{2 T_s(t,x)}} \tag{3.11}$$

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

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

$$\langle D_v \rangle V_M + \frac{\langle D_v' \rangle}{m_s} T_M = \langle v D_v \rangle \tag{3.12}$$

$$\langle v D_v \rangle V_M + \frac{\langle (v D_v)' \rangle}{m_s} T_M = \langle v^2 D_v \rangle \tag{3.13}$$

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

$$\langle G \rangle = \int_{\mathbb{R}} G f_s(t,x,v)\,dv \tag{3.14}$$

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

$$S_s(t,x,v) = S_{s,w_1} + S_{s,w_2} + S_{s,k} \tag{3.15}$$

S_{s,w_1} is a sink term describing the loss of particles in the wall. This operator conserves the charge. S_{s,w_2} is an additional sink term describing the loss of momentum and energy in the wall. Both sink terms are Bhatnagar-Gross-Krook (BGK) operators (Bhatnagar, Gross, and Krook 1954). $S_{s,k}$ is a kinetic source term similar to the one used in GYSELA (Y. Sarazin, V. Grandgirard, Abiteboul, et al. 2011), which describes the addition of particles and energy from the body of the plasma in order to compensate for the particles lost into the wall. Once a steady state is reached, this operator ensures that the total number of particles remains constant. The terms are defined as follows:

$$S_{s,w_1}(t,x,v) = -\nu_{s,w_1}(t,x)\mathcal{M}_{w_1}(x)[f_s(t,x,v) - g_{s,w}(n_w, T_{w_1}, v)] \tag{3.16}$$

$$S_{s,w_2}(t,x,v) = -\nu_{w_2}\mathcal{M}_{w_2}(x)\left[f_s(t,x,v) - g_{s,w}(n_s(t,x), T_{w_2}(t), v)\right] \tag{3.17}$$

$$S_{s,k}(t,x,v) = \frac{\mathcal{M}_k(x)}{\int_0^{L_x}\mathcal{M}_k(x')dx'}\frac{s_k\sqrt{m_s}}{\sqrt{2\pi T_k}}e^{-\frac{m_s v^2}{2T_k}} \tag{3.18}$$

where $\nu_{s,w_1}(t,x)$ is the adjusted amplitude of the particle sink for species s, $\mathcal{M}_z(x)$ are mask functions (with $z \in \{w_1, w_2, k\}$), $g_{s,w}(n,T,v)$ is the distribution function targeted by the sink operators in the wall region defined in equation (3.19), n_w is the target density in the wall, T_{w_1} is the target temperature in the wall, ν_{w_2} is the constant amplitude for the momentum and energy sink in the wall, $n_s(t,x)$ is the particle density defined in equation (3.4), $T_{w_2}(t)$ is the target temperature defined in equation (3.21), s_k is the magnitude of the source term and T_k is the temperature of the source.

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

$$g_{s,w}(n,T,v) = \frac{n\sqrt{m_s}}{\sqrt{2\pi T}}\exp\left(-\frac{m_s v^2}{2T}\right) \tag{3.19}$$

$$T_{w_2}(t) = \frac{1}{6}\left[\frac{\int[n_s(t,x_{Lw})v - \int v f(t,x_{Lw},v)dv]^2 f dv}{n_s(t,x_{Lw})^3} + \right. \tag{3.20}$$

$$\left. \frac{\int[n_s(t,x_{Rw})v - \int v f(t,x_{Rw},v)dv]^2 f dv}{n_s(t,x_{Rw})^3}\right] \tag{3.21}$$

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

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

equation:

$$\nu_{iw_1}(t,x) = \nu_{w_1} \tag{3.22a}$$

$$\nu_{ew_1}(t,x) = \nu_{w_1}\frac{n_i(t,x)-n_w}{n_e(t,x)-n_w} \tag{3.22b}$$

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

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

$$\mathcal{M}(x,x_L,x_R,d) = \frac{1}{2}\left[\tanh\left(\frac{x-x_L}{d}\right) - \tanh\left(\frac{x-x_R}{d}\right)\right] \tag{3.23}$$

$$\mathcal{M}_w(x) = 1 - \mathcal{M}(x,x_{Lw},x_{Rw},d_w) \tag{3.24}$$

$$\mathcal{M}_k(x) = \mathcal{M}(x,x_{Lk},x_{Rk},d_k) \tag{3.25}$$

$$\mathcal{M}_{w_2}(x) = \left[1-\mathcal{M}\left(x,\frac{x_0+4x_{Lw}}{5},\frac{x_N+4x_{Rw}}{5},\frac{x_{Lw}-x_0}{125}\right)\right]\cdot \left[1-\mathcal{M}\left(x,\frac{3x_0+2x_{Lw}}{5},\frac{3x_N+2x_{Rw}}{5},\frac{x_0+4x_{Lw}}{30}\right)\right] \tag{3.26}$$

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

The shape described by equations (3.24) - (3.26) can be seen in Figure 3.1b. This use of a mask function is a penalisation technique, as described by Paredes, H. Bufferand, Ciraolo, et al. 2014. The same mask function has previously been used in a very similar problem by Caschera, G. Dif-Pradalier, Ph. Ghendrih, et al. 2018.

The equations used in the simulation are unitless. This is achieved by normalising the variables in equations (3.1)-(3.13). Each unitless variable $\hat{V}$ is defined as follows:

$$\hat{V} = \frac{V}{V^\star} \tag{3.27}$$

where V is $\hat{V}$ expressed in a given set of units, and $V^\star$ is the normalisation parameter, as defined in table 3.1.

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

Table 3.1.: The parameters used to normalise the different variables that appear in the equations. The index $_s$ denotes the species (ion or electron). The index $_y$ denotes the region (wall or plasma). ε_0 is the permittivity of free space.

Equations (3.1) and (3.2) are therefore expressed as:

$$\left[\partial_{\hat{t}} + \frac{1}{\sqrt{\hat{m}_s}}\left(\hat{v}_s\partial_{\hat{x}} - \hat{q}_s\partial_{\hat{x}}\hat{\phi}(\hat{t},\hat{x})\partial_{\hat{v}_s}\right)\right]\hat{f}_s(\hat{t},\hat{x},\hat{v}_s) = \hat{S}_s(\hat{t},\hat{x},\hat{v}_s) + \hat{C}_s(\hat{t},\hat{x},\hat{v}_s) \tag{3.28}$$

$$\partial^2_{\hat{x}}\hat{\phi}(\hat{t},\hat{x}) = -\hat{\rho}_q(\hat{t},\hat{x}) \tag{3.29}$$

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

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

3.2.2. Conservation Laws

The equations described above are kinetic equations. By multiplying equation (3.28) by powers of v_s and integrating, conservation equations can be obtained from the kinetic equations. This gives a fluid model of our simulation which depends on the fluid density n_s (defined in equation (3.4)), the particle flux Γ_s, the Reynold's stress Π_s and the heat flux Q_s. These quantities are defined as follows:

$$\Gamma_s(t,x) = \int v_s f_s(t,x,v_s)\,dv_s \tag{3.30}$$

$$\Pi_s(t,x) = \int v_s^2 f_s(t,x,v_s)\,dv_s \tag{3.31}$$

$$Q_s(t,x) = \int \frac{1}{2}v_s^3 f_s(t,x,v_s)\,dv_s \tag{3.32}$$

The conservation equations describe the conservation of the particle density (3.33), the mean velocity (3.34), and the kinetic energy (3.35):

$$\partial_t n_s(t,x) + \frac{1}{\sqrt{m_s}}\partial_x\Gamma_s(t,x) = \int S_s(t,x,v_s) + C_{ss}(t,x,v_s)\,dv_s \tag{3.33}$$

$$\partial_t\Gamma_s(t,x) + \frac{1}{\sqrt{m_s}}\left(\partial_x\Pi_s(t,x) + q_s\partial_x\phi(t,x)n_s(t,x)\right) = \int v_s S_s(t,x,v_s) + v_s C_{ss}(t,x,v_s)\,dv_s \tag{3.34}$$

$$\partial_t\Pi_s(t,x) + 2\frac{1}{\sqrt{m_s}}\left(\partial_x Q_s(t,x) + q_s\partial_x\phi(t,x)\Gamma_s\right) = \int v_s^2 S_s(t,x,v_s) + v_s^2 C_{ss}(t,x,v_s)\,dv_s \tag{3.35}$$

These equations therefore provide a method for evaluating the error of the system. This is done by comparing numerical approximations of the left and right-hand sides of equations (3.33)-(3.35). These errors will be used in Section 3.5 to evaluate the precision of the numerical schemes.

3.3. Semi-Lagrangian Scheme

In this section the main methods used to resolve the system defined by equations (3.28) and (3.29) will be described. Namely the time stepping method, the source and sink operators, the collision operator, the semi-Lagrangian advection, and the Poisson solver. Three of these methods are based on splines.

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

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

3.3.1. Time Stepping

The system described by equations (3.28) and (3.29) is solved using a predictor-corrector scheme. Strang's second order splitting scheme (Strang 1968) is used to calculate $\partial_t f_{s,n} = \partial_t f_s(t_n, x, v_s)$ at each step of the scheme. Equation (3.28) is therefore broken up into six equations: a spatial advection equation (3.36), a velocity advection equation (3.37), two equations describing the sinks (3.38),(3.39), an equation describing the source (3.40), and an equation describing the collisions (3.41):

$$\partial_t f_{s,n} = -\frac{1}{\sqrt{m_s}} v_s \partial_x f_{s,n} \tag{3.36}$$

$$\partial_t f_{s,n} = \frac{q_s}{\sqrt{m_s}} \partial_x \phi_n \partial_{v_s} f_{s,n} \tag{3.37}$$

$$\partial_t f_{s,n} = -\nu_{s,w_1}(t,x)\mathcal{M}_{w_1}(x)(f_{s,n} - g_{s,w}(n_w, T_{w_1}, v_s)) \tag{3.38}$$

$$\partial_t f_{s,n} = -\nu_{w_2}\mathcal{M}_d(x)(f_{s,n} - g_{s,d}(n_s(t,x), T_{w_2}(t), v)) \tag{3.39}$$

$$\partial_t f_{s,n} = \frac{\mathcal{M}_k(x)}{\int_{x_0}^{x_{Nx-1}} \mathcal{M}_k(x')dx'} \frac{s_k}{\sqrt{2\pi T_k}} e^{-\frac{v_s^2}{2T_k}} \tag{3.40}$$

$$\partial_t f_{s,n} = \partial_{v_s}\left(D_{v_s}(t,x,v_s)\,\partial_{v_s} f_{s,n} + D_{v_s}(t,x,v_s)\frac{(v_s - V_M(t,x))}{T_M(t,x)} f_{s,n}\right) \tag{3.41}$$

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

Equation (3.37) is chosen as the central operator of the Strang splitting.

Let us define f_n the vector $\{f_{i,n}, f_{e,n}\}$, and six operators : $\mathcal{X}_{\Delta t}$, $\mathcal{V}_{\Delta t}(\phi_n)$, $\mathcal{W}_{\Delta t}$, $\mathcal{D}_{\Delta t}$, $\mathcal{K}_{\Delta t}$, $\mathcal{C}_{\Delta t}$ which are defined as the operators which solve the six equations (3.36)-(3.41) for $f_{i,n}$, then $f_{e,n}$.

The operator $\mathcal{A}_{\Delta t}$ which solves equation (3.28) is therefore defined as:

$$\mathcal{A}_{\Delta t}(f_n, \phi_n) = \left(\mathcal{W}_{\frac{\Delta t}{2}}\mathcal{C}_{\frac{\Delta t}{2}}\mathcal{K}_{\frac{\Delta t}{2}}\mathcal{D}_{\frac{\Delta t}{2}}\mathcal{X}_{\frac{\Delta t}{2}}\mathcal{V}_{\Delta t}(\phi_n)\mathcal{X}_{\frac{\Delta t}{2}}\mathcal{D}_{\frac{\Delta t}{2}}\mathcal{K}_{\frac{\Delta t}{2}}\mathcal{C}_{\frac{\Delta t}{2}}\mathcal{W}_{\frac{\Delta t}{2}}\right) f_n \tag{3.42}$$

The second-order time stepping algorithm, which approximates the solution to the

system described by equations (3.28) and (3.29), can finally be summarised as follows:

$$f_{n+1} = \mathcal{A}_{\Delta t}\left(f_n, \mathcal{P}\mathcal{A}_{\frac{\Delta t}{2}} f_n\right) \tag{3.43}$$

$$\phi_{n+1} = \mathcal{P}\left(f_{n+1}\right) \tag{3.44}$$

where $\mathcal{P}$ is the Poisson operator described in Section 3.3.3.

3.3.2. Vlasov

The advection equations (3.36) and (3.37) are solved using a backward semi-Lagrangian scheme. This scheme is a mixture between the well-known PIC and Eulerian methods (Sonnendrücker, Roche, Bertrand, et al. 1999). The trajectories that particles located at each grid point would have followed, are traced back to find the location of the particles at the previous time step. These trajectories are known as characteristics and the location at the previous time step is known as the foot of the characteristic. During a simple advection, the value of the distribution function remains constant along a characteristic; therefore the value at the grid point is equal to the value at the foot of the characteristic. Figure 3.2 illustrates the location of the foot of the characteristic $x_6^\star$ for a grid point x_6. For constant advection the foot of the characteristic can be found trivially.

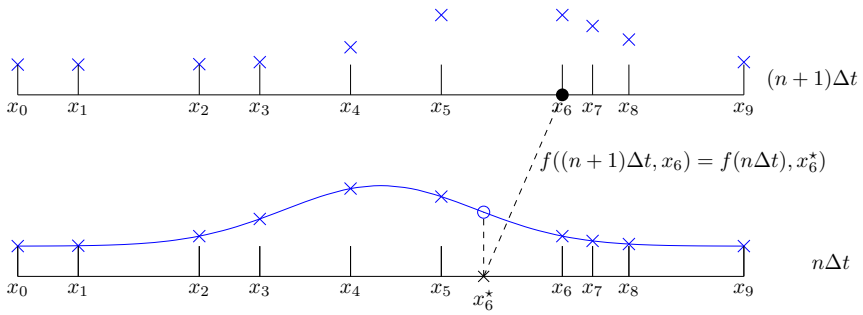


Figure 3.2.: A visual description of the semi-Lagrangian scheme where $x_6^\star$ is the foot of the characteristic of x_6.

As seen in Figure 3.2, the foot of the characteristic is not necessarily found at a grid point. Therefore an interpolation method is required to approximate the value at this point. In this work, a spline interpolation method is used. This is a common choice used in many codes (Manfredi, Hirstoaga, and Devaux 2010; Afeyan, Casas, Nicolas Crouseilles, et al. 2014), including the GYSELA code (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016) The values at the previous time step are used to construct a spline representation of the function. The spline representation can then be evaluated at any point on the domain as described in Section 2.6. It is evaluated at the feet to provide the updated value of the function at the grid point.

The distribution function is assumed to be constant outside of the domain. Therefore if the foot of the characteristic falls outside the domain, the boundary value is

used to update the value of the function at the grid point. This is justified by the fact that the distribution function falls to zero in the wall and tends to zero for large velocity magnitudes.

3.3.3. Poisson

The quasi-neutrality condition defined in equation (3.29) takes the form of a Poisson equation. It is solved using the FEM constructed on a spline basis. Most other codes, including the GYSELA code use alternative methods such as Fourier transforms (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016) or FDM (G. Manfredi and Valsaque 2004; Manfredi, Hirstoaga, and Devaux 2010). The choice was made to use FEM instead of FDM as it is simpler to express the equations on non-uniform points. FEM is preferred over Fourier transforms as the latter place unnecessary restrictions on the geometry of the problem. As GYSELA will be extended in the future to include more complex geometries it is important to test methods which do not constrain these choices.

The spline basis used for the FEM is a spline basis with all boundary knots placed on the boundary. We use the notation $\{b_{0,d}(x), \dots, b_{n_b-1,d}(x)\}$ to denote the spline basis for splines of degree d, where n_b is the number of basis splines. Greville boundary conditions are used in the simulation, so that n_b is equal to the number of interpolation points (N_x in the spatial direction, and N_v in the velocity direction).

The approximations of the electric potential (3.45) and the charge density (3.46) are expressed on the spline basis as follows:

$$\tilde{\phi}_n(x) = \sum_{j=0}^{n_b-1} b_{j,d}(x)\, C_{\phi_j} \tag{3.45}$$

$$\tilde{\rho}_{n,q}(x) = \sum_{j=0}^{n_b-1} b_{j,d}(x)\, C_{\rho_{q,j}} \tag{3.46}$$

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

This leads to the following expression for the weak form of equation (3.29):

$$\sum_{j=0}^{n_b-1} \left[\partial_x b_{i,d}(x) b_{j,d}(x) \Big|_{x_0}^{x_{N_x-1}} - \int_{x_0}^{x_{N_x-1}} \partial_x b_{i,d}(x) \partial_x b_{j,d}(x)\, dx \right] DC_{\phi_j} \tag{3.47}$$

$$= \sum_{l=0}^{n_b-1} C_{\rho_{q,l}} \int_{x_0}^{x_{N_x-1}} b_{i,d}(x) b_{l,d}(x)\, dx \tag{3.48}$$

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

Dirichlet boundary conditions are imposed on the electric potential:

$$\sum_{j=0}^{n_b-1} b_{j,d}(x_0)\phi_j = \sum_{j=0}^{n_b-1} b_{j,d}(x_{N_x-1})\phi_j = 0 \tag{3.49}$$

As shown in Figure 2.2, the non-uniform spline basis is defined such that the subset $\{b_1^d, \dots, b_{n_b-2}^d\}$ is a basis for splines respecting the Dirichlet condition. Equation (3.48) can therefore be simplified to:

$$-\sum_{j=1}^{n_b-2} \int_{x_0}^{x_{N_x-1}} \partial_x b_{i,d}(x) \partial_x b_{j,d}(x)\, dx\, C_{\phi_j} = \sum_{l=0}^{n_b-1} C_{\rho_{q,l}} \int_{x_0}^{x_{N_x-1}} b_{i,d}(x) b_{l,d}(x)\, dx \tag{3.50}$$

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

By allowing the index i to vary over all functions in the reduced basis, equation (3.50) leads to a matrix equation of the form $S\vec{\phi} = \vec{r}$, where each of the elements of the matrix S and the vector $\vec{r}$ are defined as:

$$S_{ij} = -\int_{x_0}^{x_{N_x-1}} \partial_x b_{i,d}(x) \partial_x b_{j,d}(x)\, dx \tag{3.51}$$

$$r_i = \sum_{l=0}^{n_b-1} \rho_{q,l} \int_{x_0}^{x_{N_x-1}} b_{i,d}(x) b_{l,d}(x)\, dx \tag{3.52}$$

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

There are several possible ways of calculating the integrals in equation (3.50). Different options including Gauss-Legendre quadrature, integration by parts, and recursive solutions using the properties of B-Splines were explored by Vermeulen et al. (Vermeulen, Bartels, and Heppler 1992). In this work we choose Gauss-Legendre quadrature as the sample points can be chosen such that the product of two splines of degree d is integrated exactly. This is achieved by using $d+1$ sample points per cell. Thus no additional error is introduced unnecessarily.

Using equation (3.3), the charge density $\rho_{q,n}$ is defined as:

$$\rho_{q,n}(t,x) = \int f_{i,n}(t,x,v_i)\, dv_i - \int f_{n,e}(t,x,v_e)\, dv_e \tag{3.53}$$

This integral does not describe a product of splines so the options suggested by Vermeulen et al. (Vermeulen, Bartels, and Heppler 1992) are not adapted to this situation. It would be possible to use these methods by first approximating the functions with a spline, however the calculation of the spline coefficients is a costly operation. To avoid this unnecessary cost, a quadrature method is used to calculate the charge density and all other integrals which don't involve splines. The chosen quadrature method is described in Section 2.8.2.1 and is derived from spline integration.

3.3.4. BGK Wall

The wall is described by equations (3.38) and (3.39), which are BGK operators. In the case of equation (3.38), the coefficients ν_{s,w_1}, defined in equation (3.22), are time-dependent, while in the case of equation (3.39), it is the function $g_{s,w}(n_s(t,x), T_d(t), v)$

which is time-dependent. These are non-standard versions of the operator which require special handling, as it means that it is not possible to solve equations (3.38) and (3.39) exactly. Instead, a second-order explicit Runge-Kutta scheme, known as the midpoint method is used. This method is defined as follows:

$$f_{s,n+\frac{1}{2}} = f_{s,n} + \frac{\Delta t}{2}\partial_t f_{s,n} \tag{3.54a}$$

$$f_{s,n+1} = f_{s,n} + \Delta t \partial_t f_{s,n+\frac{1}{2}} \tag{3.54b}$$

where $f_{s,n} = f_s(t_n,\cdot,\cdot)$ is the approximation of the distribution function at time t_n , Δt is the time step, and $\partial_t f_{s,n}$ is given by equation (3.38) or (3.39).

Let us define the operator $\nu\left(f_n\right)$ which calculates the normalised frequencies given the approximations of the distribution functions at a given time t_n using equations (3.4) and (3.22):

$$\nu\left(f_n\right) = \begin{cases} \nu_{i,w_1} = \nu_{w_1} \\ \nu_{e,w_1} = \nu_{w_1}\frac{\int f_{i,n}(x,v_i)dv_i - n_w}{\int f_{e,n}(x,v_e)dv_e - n_w} \end{cases} \tag{3.55}$$

The integrals required for this calculation are the same as those in equation (3.53) and will therefore be solved in the same way; namely using the spline quadrature scheme described in Section 2.8.2.1.

Once the normalised frequencies have been calculated, equations (3.38) and (3.39) can be solved analytically.

Let us define the operator $\mathcal{W}^*_{\Delta t}\left(f_n,\nu_s\right)$ which resolves equation (3.38) for known frequencies. The second-order operator $\mathcal{W}_{\Delta t}\left(f_n\right)$ is therefore defined as:

$$\mathcal{W}_{\Delta t}\left(f_n\right) = \mathcal{W}^*_{\Delta t}\left(f_n, \nu\left\{\mathcal{W}^*_{\frac{\Delta t}{2}}\left(f_n,\nu\left\{f_n\right\}\right)\right\}\right) \tag{3.56}$$

Similarly, by defining the operator $\mathcal{D}^*_{\Delta t}\left(f_n,g_{s,d}\right)$ which resolves equation (3.39) for a given density and temperature, we obtain:

$$\mathcal{D}_{\Delta t}\left(f_n\right) = \mathcal{D}^*_{\Delta t}\left(f_n, g\left\{\mathcal{D}^*_{\frac{\Delta t}{2}}\left(f_n,g\left\{f_n\right\}\right)\right\}\right) \tag{3.57}$$

3.3.5. Kinetic Source

The particle source evolution is defined in equation (3.40). This equation is sufficiently simple to allow it to be solved analytically. None of the terms depend on time or on the distribution function. Therefore the operator $\mathcal{K}_{\Delta t}$ is defined simply as:

$$\mathcal{K}_{\Delta t}(f_{s,n}) = f_{s,n} + \Delta t \frac{\mathcal{M}_k(x)}{\int_{x_0}^{x_{N_x-1}}\mathcal{M}_k(x')dx'}\frac{s_k}{\sqrt{2\pi T_k}}e^{-\frac{v_s^2}{2T_k}} \tag{3.58}$$

3.3.6. Collisions

The intra-species collisions are described by equation (3.41). By expressing the distribution function on the spline basis, this equation can be rewritten as:

$$\partial_t f_{s,n} = \sum_{i=0} \Big[\partial_v D_v(x_j,v)\,\partial_v b_{i,d}(v) + D_v(x_j,v)\,\partial_v^2 b_{i,d}(v) + \partial_v D_v(x_j,v)\frac{m_a(v-V_M(x_j))}{T_M(x_j)} b_{i,d}(v) + D_v(x_j,v)\frac{m_a}{T_M(x_j)} b_{i,d}(v) + D_v(x_j,v)\frac{m_a(v-V_M(x_j))}{T_M(x_j)}\partial_v b_{i,d}(v)\Big] C_{f^n_{i,j}} \quad (3.59)$$

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

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

$$V_M((n+1)\Delta t, x_j, v) \approx V_M(n\Delta t, x_j, v) \quad (3.60)$$
$$T_M((n+1)\Delta t, x_j, v) \approx T_M(n\Delta t, x_j, v) \quad (3.61)$$
$$D_v((n+1)\Delta t, x_j, v) \approx D_v(n\Delta t, x_j, v) \quad (3.62)$$

The derivative $\partial_v D_v(x,v)$ can be calculated analytically using the definition of $D_v(x,v)$. The integrals necessary to define the temperature $T_s(t,x)$ and average velocity $V_s(t,x)$ are calculated similarly to the density $n_s(t,x)$ using the spline quadrature described in Section 2.8.

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

$$\frac{f_s\left((n+1)\Delta t, x_j, v\right) - f_s\left(n\Delta t, x_j, v\right)}{\Delta t} = \frac{1}{2}\left[\mathcal{C}_{ss}((n+1)\Delta t, x_j, v) + \mathcal{C}_{ss}(n\Delta t, x_j, v)\right] \quad (3.63)$$

Combining equations (3.63) and (3.59) allows us to express the scheme as a matrix equation of the form:

$$(I - M) C_{f_j^{n+1}} = (I + M) C_{f_j^n} \quad (3.64)$$

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

$$M_{li} = \frac{\Delta t}{2}\Bigg\{ \left[\partial_v D_v(x_j,v_l)(v_l - V_M(x_j)) + D_v(x_j,v_l)\right]\frac{m_a}{T_M(x_j)} b_{i,d}(v_l) + \left(\partial_v D_v(x_j,v_l) + D_v(x_j,v_l)\frac{m_a(v_l - V_M(x_j))}{T_M(x_j)}\right)\partial_v b_{i,d}(v_l) + D_v(x_j,v_l)\,\partial_v^2 b_{i,d}(v_l)\Bigg\} \quad (3.65)$$

3.3.7. Conservation

The error in the conservation equations (3.33) - (3.35) is calculated by approximating each of the terms using the calculated distribution function and comparing the left and right-hand side of the equations.

All derivatives in these equations are calculated using spline interpolation. All integrals in these equations are calculated using the spline quadrature described in Section 2.8.2.1.

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

Equations (3.30) - (3.35) contain integrals which are calculated on the domain $[-\infty,\infty]$, however a numerical representation cannot capture the entire domain. This means that the choice of grid can influence the conservation error. A larger domain in the velocity dimension will lead to less potential error in the conservations. Despite this, it is not necessary to have a very large domain in the velocity dimension. The distribution function can be approximated by a Maxwellian function in the velocity direction. Maxwellians can be truncated at relatively low values without causing large errors in the integrals. This is detailed in A. For our simulations we will use the domain $[-6,6]$ which would lead to an error of $4.95\cdot10^{-9}$ for a Maxwellian function centred on 0.

3.4. Non-Uniform Convergence results

There are many different ways to define a set of non-uniform points and different choices for these points can lead to different convergence behaviour. The method used in this work is based on the following weight function (Farrashkhalvat and Miles 2003):

$$W(x) = \sqrt{1+\alpha^2\left(u_x(x)\right)^2} \tag{3.66}$$

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

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

$$u_x(x) = L_x \max_{t,v}\left|\partial_x f(t,x,v)\right| \tag{3.67}$$

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

are inverted.

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

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

$$\hat{W}(x) = \frac{W(x) - 1}{\max_x [W(x) - 1]} \tag{3.68}$$

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

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

An alternative method for calculating the non-equidistant points from the weight function is the equipartition method. In this method the points $[x_0 < \cdots < x_N]$ are placed such that the grid : $[W(x_0), \ldots, W(x_N)]$ is equidistant. The difficulty with this method is the need to know the weight function precisely. This means that any change to the simulation parameters requires another test simulation to be run with uniform points. If the simulation becomes so large that even with the maximum number of uniform points (dependent upon the available memory) NaN values appear, then a non-uniform simulation with points chosen using the equipartition method would not be possible. On the other hand, by defining the points such that they are uniform by zone, it is simple to estimate the point density required for larger simulations. For example, if the simulation domain is increased such that the plasma takes up an additional distance d, then the point density can be maintained, but the size of the two zones between the plasma source and the wall should each be increased by $d/2$. Similarly if the mask describing the wall is made twice as sharp, the point density in this region can be doubled.

The chosen cells are then used as the cells of the splines. The points on which the simulation evolves are the interpolation points of the splines, defined as the Greville abcissae (see equation (2.46)).

3.4.1. Splines

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

$$f(x) = \frac{1}{2\pi} \exp\left(-\frac{x^2}{2}\right) \tag{3.69}$$

A Maxwellian function is an approximation of the distribution of particles in velocity space in a plasma, it is therefore an interesting test case when examining the convergence. In the simulations the domain $[-6,6]$ will be used, however in this section the domain $[-30,30]$ is considered. A larger domain is considered here as the gradients of a Maxwellian function in the domain $[-6,6]$ are small compared to the size of the domain. This leads to quasi-equidistant points. This is not the case in the simulation, as larger gradients arise due to the interaction between the plasma and the wall as we will see in Section 3.5.

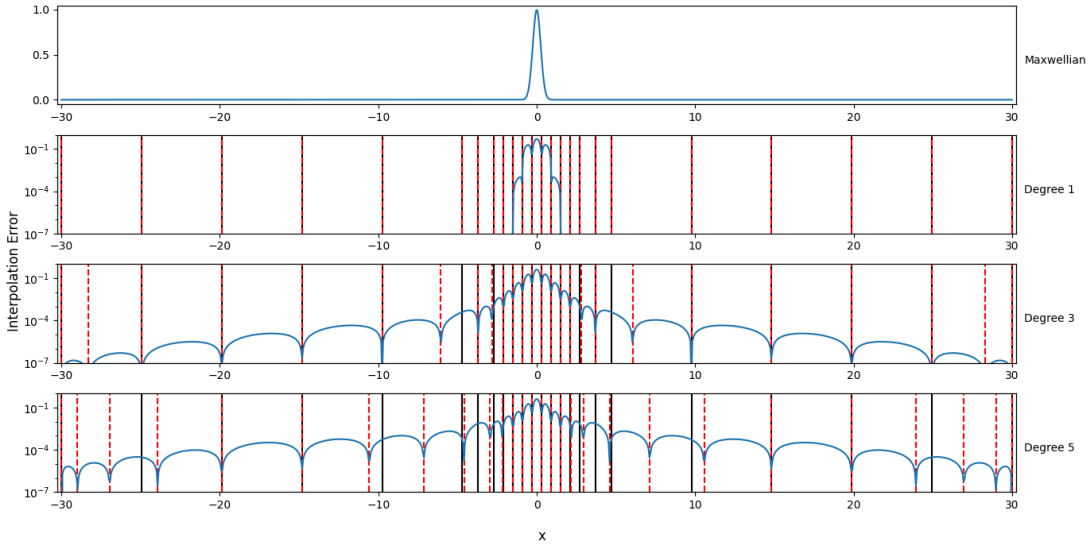


Figure 3.3.: The error for the spline interpolation of equation (3.69), for splines of varying degrees. The knots of the spline are shown in black, while the interpolation points are shown by red dashed lines.

Figure 3.3 shows the error of the spline approximation of the function defined in equation (3.69) for various degrees of splines. The knots and Greville points are shown on the figure. We can see that the main factor of influence on the error is the proximity to an interpolation point. We also note that the L-∞ error decreases as the degree of the spline increases, however using an overly high ordered spline is not necessarily advantageous as they can introduce small spurious oscillations in areas with little variation. In Figure 3.3, this can be seen near the boundary where the error increases with the spline degree.

A spline of degree d is said to be of order $d+1$, it is therefore important to confirm that the order agrees with our expectations. Figure 3.4 shows the convergence of

the L2 norm of the interpolation error as a function of the number of cells. We see that the order increases as the degree increases, and that it is close to the expected value. However the non-equidistant points do have an effect on the convergence order. The order of convergence is slightly lower in the non-uniform case. Despite this, the improved choice of points leads to a smaller error unless a very large number of points are used.

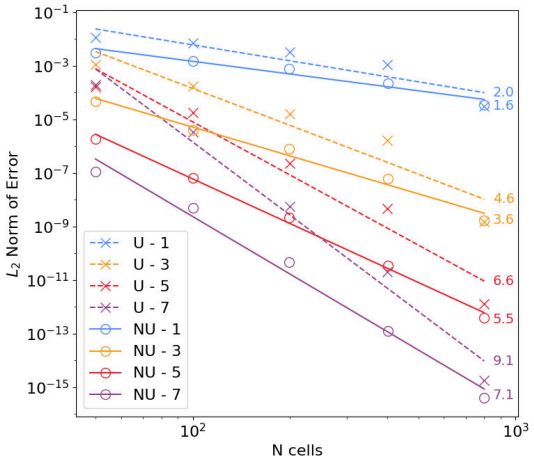


Figure 3.4.: The convergence of the L2 norm of the interpolation error for a spline interpolation of a Maxwellian function for different degrees. "U - d" indicates uniform splines of degree d, while "NU - d" indicates non-uniform splines of degree d.

Given that the error decreases rapidly for higher order splines it seems natural to use higher order splines in our simulations. However there is a trade-off between the error that can be obtained from the spline approximation and the time required to carry out the calculation. One way to reduce the temporal cost associated with higher order splines is to use effective parallelisation techniques. GPUs can allow us to carry out a large number of similar calculations in parallel. The use of GPUs for Vlasov equations solved using a backward semi-Lagrangian method on a spline basis with equidistant knots has already been studied by multiple groups (Mehrenberger, M., Steiner, C., Marradi, L., et al. 2013; Guillaume Latu 2011). However as discussed in Section 2.6, non-uniform splines are less performant and may therefore potentially benefit more from the use of GPUs. This point will be discussed in Section 3.4.2.

3.4.2. Advection

There are two potential sources of errors for semi-Lagrangian advection: the calculation of the foot of the characteristic, and the evaluation of the function at that point. In this work we consider constant advection which allows an exact calculation of the foot of the characteristic. The error is therefore entirely described by the spline error.

Figure 3.3 shows the error over the domain. We can see that the advection error will be smallest when the step is such that the feet of the characteristics are close to the knots.

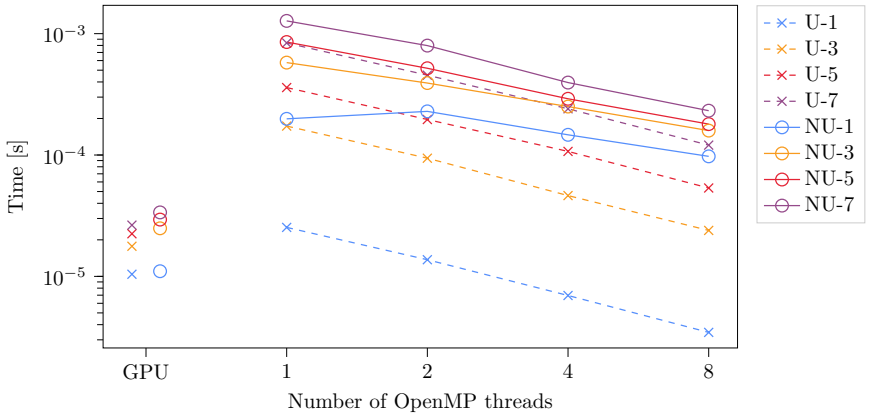


Figure 3.5.: The time required to run an advection step for a grid with 2048 grid points for uniform (x) and non-uniform (∘) splines of various degrees on code accelerated with OpenMP for multi-threading or OpenACC for GPUs. Tests were run at the Centre de Calcul Intensif d'Aix Marseille.

As constant semi-Lagrangian advection consists only of spline interpolation and spline evaluation, the advection module can be used to examine the performance of the splines. Figure 3.5 shows the time taken for each advection step of the simulation for different parallelisation methods and spline degrees, for uniform and non-uniform splines. The OpenMP tests were run on a SkyLake processor with 192 GB of RAM. The OpenACC tests were run on a NVIDIA V100 GPU with 380GB of RAM. As expected, we note that non-uniform splines are significantly more costly than uniform splines, and that higher degree splines are also more costly. However we can also see that the scheme scales well when it is parallelised using OpenMP. This can allow the cost to be attenuated somewhat. We also see that GPUs present themselves as the natural solution to this problem. When using GPUs the increased cost is minimal and the total time required to run even potentially heavy simulations using non-uniform splines of degree 7 is still not significantly more costly than running uniform splines of degree 1 in serial.

3.4.3. Poisson

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

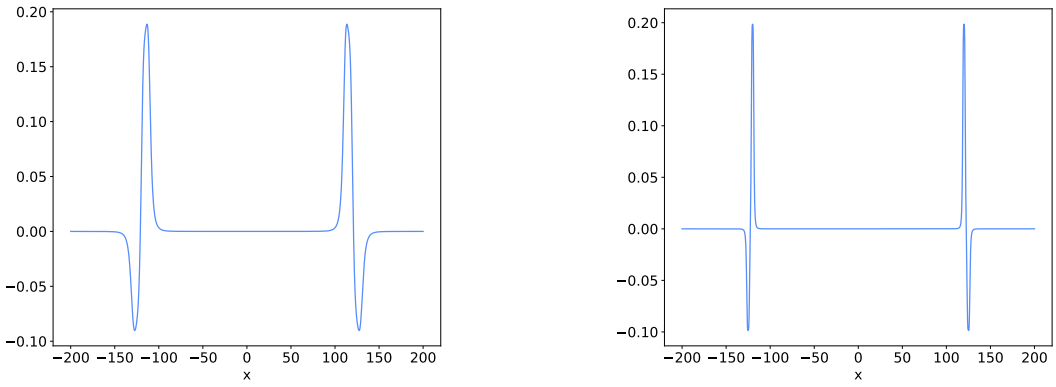
of the equation is defined as:

$$G(x, x0, w) = \frac{1}{w + (x - x0)^2} \tag{3.70}$$

$$\rho_1(x) = 200\,[G(x, -115, 1000) - G(x, -125, 2000) + G(x, 115, 1000) - G(x, 125, 2000)] \tag{3.71}$$

$$\rho_2(x) = G(x, -120, 5) - G(x, -125, 10) + G(x, 120, 5) - G(x, 125, 10) \tag{3.72}$$

where $G(x, x0, w)$ is a Lorentzian function, and $x \in [-200, 200]$. This choice provides an equation with an analytical solution. The shape of $\rho_1(x)$ and $\rho_2(x)$ is chosen to be similar to the expected charge density profile (see Section 3.5). The shape can be seen in Figure 3.6.



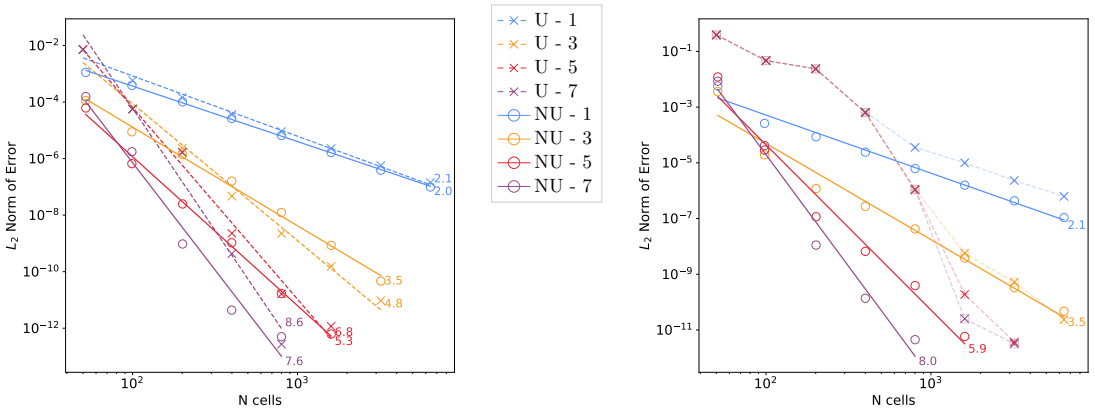
(a) Equation (3.71). (b) Equation (3.72).

Figure 3.6.: The right-hand side of the test equation for the non-uniform Poisson solver.

The results are shown in Figure 3.7. In Figure 3.7a, we see that once again there is some loss of convergence order compared to the theoretical value due to the use of non-equidistant knots. However the error still improves as the degree of the spline increases and machine precision is reached rapidly for higher order splines. In contrast, in Figure 3.7b we see that uniform points struggle to handle very steep gradients. The order of convergence cannot be calculated for these points. For the first few points we see a superposition of the calculated error. This occurs because there are not enough points at the peaks to accurately describe their shape. Thus, until there is a sufficient number of points, the convergence describes the sampling issues, not the Poisson scheme. On the other hand non-uniform points have no problem with such steepness as there is a higher point density in this area. These points therefore converge with the expected order.

m_i	400	m_e	1	q_i	1	q_e	-1	n_w	10^{-11}	T_w	0.5
ν_w	0.1	ν_d	0.1	s_k	0.1	T_k	1	ν_{ii}	0.1	ν_{ee}	0.1
x_{Lw}	147	x_{Rw}	553	x_{Lk}	322	x_{Rk}	378	d_w	0.1	d_k	20
Δt	0.1	x_0	0	x_{N_x-1}	700	v_0	-6	v_{N_v-1}	6		

Table 3.2.: Definition of the constants used in the simulation described in Section 3.2.



(a) Equation (3.71). (b) Equation (3.72).

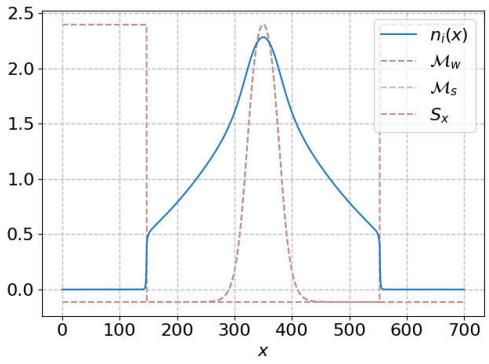
Figure 3.7.: The convergence of the L2 norm of the error when solving equations (3.71) and (3.72) with the uniform (x) and non-uniform (∘) Poisson solver for different degrees of splines.

3.5. Plasma Simulations with Non-Equidistant Meshes

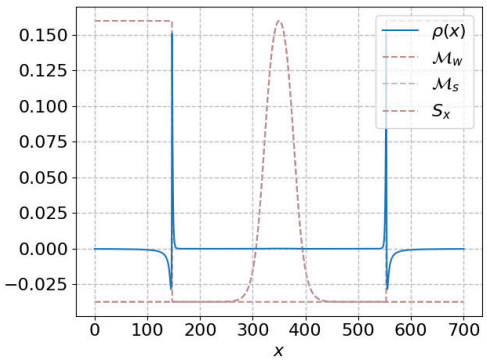
The tools described in sections 3.3 and 3.4 will now be used to run the simulation described in Section 3.2. The code used for this simulation is known as VOICE (Vlasov Open boundary Ion Coupling to Electrons). The domain used for the simulation is $[0,700] \times [-6,6]$. The parameters are described in table 3.2. The simulation was run for 20 000 time steps to a final time of $T = 2000$.

First we show some basic physical results showing that our simulation acts as expected. In figure 3.8a we see that the plasma is correctly redistributed. The initial conditions describe a plasma with a homogeneous density throughout the domain; but the plasma density in the wall rapidly falls to zero as an equilibrium is established between particles injected by the source and particles absorbed by the wall. A higher density is seen in the central region where the particle source is placed.

In figure 3.8b we see the charge density profile. The quasi-neutrality is respected in the plasma region, but a positive charge develops in front of the wall forming the sheath region. This shows a build-up of ions approaching the wall which have been accelerated by the sheath. Electrons in the sheath are accelerated towards the plasma,



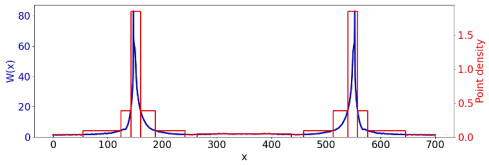
(a) Ion density as defined in equation (3.4).



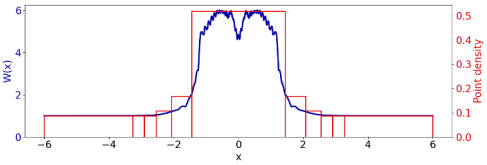
(b) Charge density as defined in equation (3.3).

Figure 3.8.: Physical results of a simulation with 2500 non-uniform points in the spatial dimension, 2000 non-uniform points in the velocity dimension at time t=2000, and splines of degree 3.

however fast electrons are not sufficiently affected to change their trajectory, as a result we also see a build-up of electrons inside the wall which have been slowed as they exit the plasma region. These electrons have not yet been absorbed in order to preserve total charge. This is a direct consequence of the charge conservation imposed by equation (3.22) and explains the negative charge seen inside the wall. This equilibrium is discussed by Munschy, Bourne, Guilhem Dif-Pradalier, et al. 2022 in a paper which focuses on the physical system.



(a) Spatial dimension.



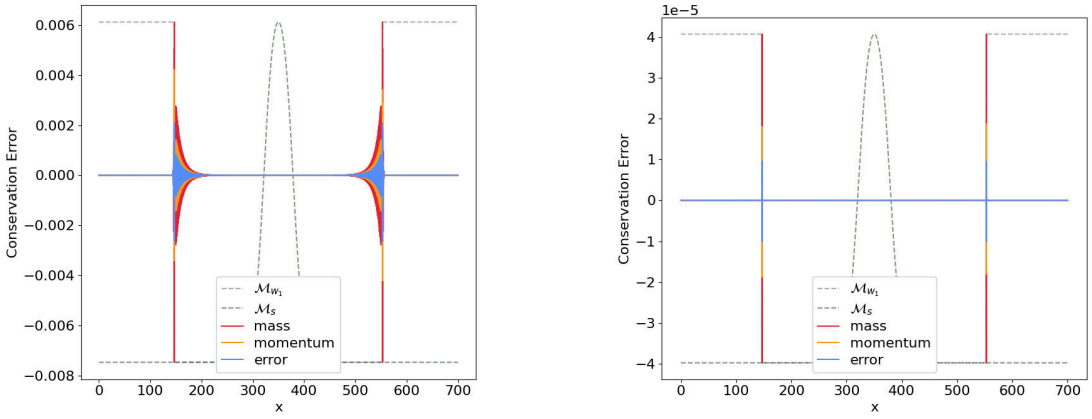
(b) Velocity dimension.

Figure 3.9.: The weight function in each dimension calculated from a reference simulation with 1500 equidistant points in the spatial dimension, 500 equidistant points in the velocity dimension, and splines of degree 3, and the point density inferred from that function.

The non-uniform points chosen were calculated from a reference simulation run with equidistant points. This simulation was run with 1500 equidistant points in the spatial dimension and 500 equidistant points in the velocity dimension. The number of points used for this simulation was kept to a minimum, however the number of points in the spatial dimension is still relatively high. This is because the steep wall creates oscillations in under-resolved simulations which eventually lead to the appearance of NaN values in the simulation. The weight functions calculated for this reference simulation can be seen in Figure 3.9.

As mentioned in Section 3.2.2 equations (3.33)-(3.35) are used to provide a method for evaluating the error which is decoupled from the equations used to solve the system. Here we only consider the error for the ions. Electrons move much faster so their resolution is time-limited. This makes it very costly to run simulations which would allow us to observe convergence in the spatial dimension. The different terms in the conservation equations can be seen in Figure 3.10. We see that spatial derivatives close to the steep gradients describing the wall oscillate unphysically and the value at the plasma-wall transition is extremely large.

The error associated with these equations is shown in Figure 3.11a. The error is concentrated around the wall transition region, with additional errors propagated into the plasma region due to oscillations. Non-equidistant points can be used to improve these errors.



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

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

Figure 3.11.: The error associated with the conservation of density (equation (3.33)), velocity (equation (3.34)), and energy (equation (3.35)) for ions at time t=2000 with splines of degree 3.

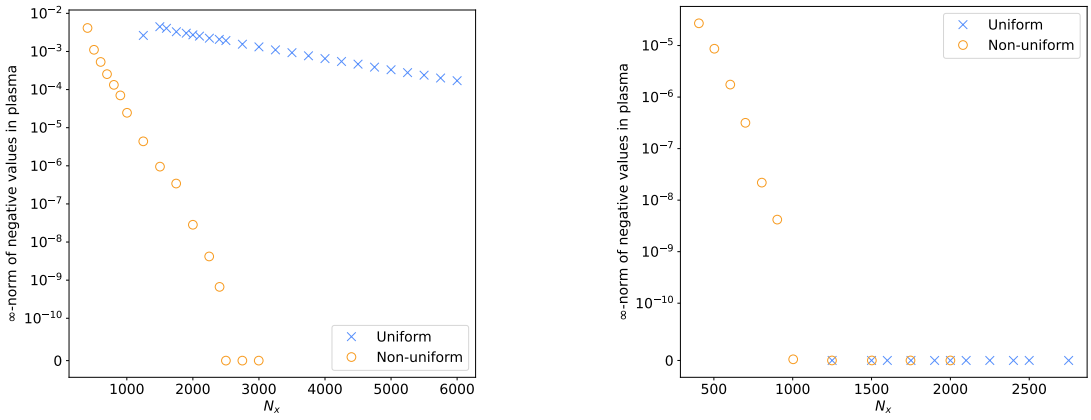
Figure 3.12 shows the variation of the conservation errors as the number of uniform or non-uniform points in the spatial dimension is varied. We see that the error converges much more rapidly when non-uniform points are used, rapidly reaching an error equivalent to the truncation error discussed in Section A. The conservation errors for a case refined with non-equidistant points can be seen in Figure 3.11b. We see that the large unphysical oscillations have been removed from the simulation.

The conservation represents fluid quantities, however the code described here is a kinetic code. Such codes contain more details which can be hard to model. We will therefore also examine a second non-fluid criteria: the presence of negative values in the plasma. A distribution function can never be negative, however our numerical methods do not impose the positivity. Spurious oscillations caused by steep gradients

can easily lead to the presence of negative values. Figure 3.13 shows the convergence of the negative values in the plasma. The plasma region is identified as the central region between 2 points identifying the start of the wall. The start of the wall is defined as the place where the charge density is equal to zero. This is calculated for the most refined non-uniform case and the same positions are used to identify the region for all other cases.

The simulation can only be used for physical studies when there are no more negative values in the plasma region. We see that for non-uniform points this occurs at Nx=2500. However the convergence for uniform points is much slower. It was not feasible to run a large enough simulation to remove all negative points while using equidistant points, however if it is assumed that the necessary spatial resolution would be equal to the smallest resolution in the successful non-equidistant case, we calculate that 23 217 points would be necessary. This is more than 9 times as many points as in the non-uniform case.

In Section 3.4 we showed that higher order splines can lead to lower errors and that the cost of using these splines is minimal when using GPU acceleration. It is therefore logical to try to use higher order splines in our simulation. Table 3.3 shows the results for a case with 1000 cells in the spatial dimension and 503 cells in the velocity dimension. Tests cannot be run with degree 1 splines as the collisions use the second derivative of the spline so the problem would not be fully defined. We see that the L_2 norm of the conservation errors decreases as the spline order increases from 3 to 5 however the difference stagnates as we increase to degree 7. This is due to the oscillations that are introduced by high order splines (as seen in Figure 3.3).



(a) Ions. (b) Electrons.

Figure 3.13.: The magnitude of the most negative values in the distribution function restricted to the plasma region for simulations with 2000 points in the velocity dimension and splines of degree 3.

For negative values in the plasma, the higher degree introduces oscillations which quickly cause problems, as can be seen in table 3.4. Negative values appear in the plasma region of both the ion and electron distribution function for degrees larger than

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

Table 3.3.: Comparison of conservation errors for different degrees of splines for a simulation with 1000 cells in the spatial dimension and 503 cells in the velocity dimension.

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

Table 3.4.: Comparison of negative values in the distribution function for different degrees and different species.

3. For electrons the convergence usually occurs more quickly as they move towards a stable configuration more quickly. This can be seen in Figure 3.13b. However this is not the case for higher order splines. As a result more points are required to respect the positivity of the distribution function when higher order splines are used.

3.6. Conclusion

In this chapter we have shown that a judicious choice of non-uniform points can be used to reduce the memory constraints for sheath simulations with steep gradients by 89%. Without this reduction it would be impossible to run the equivalent simulation with uniform points on a GPU node of the Centre de Calcul Intensif d'Aix Marseille (380GB of RAM). These improvements have therefore permitted the physical study conducted by Munschy, Bourne, Guilhem Dif-Pradalier, et al. 2022. Furthermore although these are already large simulations, they do not yet represent the real dimensions of the domain we are simulating. These improvements will allow the simulation domain to be extended to simulate a situation closer to reality.

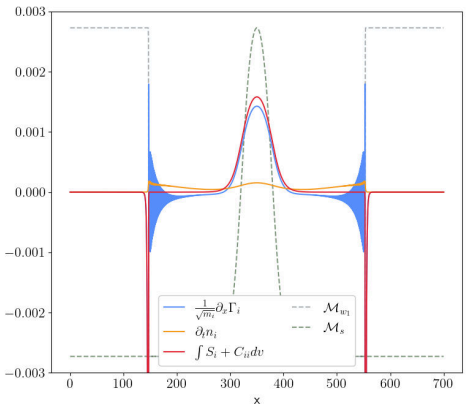
Different degrees of spline were compared for the main numerical schemes used in the simulation. It was found that high-order schemes converge faster but can introduce spurious oscillations. This means that high-order schemes are not adapted to kinetic simulations as more points are required to ensure that the distribution

function remains positive. Cubic splines seem to be a good compromise, as they converge quickly without adding too many oscillations, which allows the distribution function to preserve its positivity with a limited number of points.

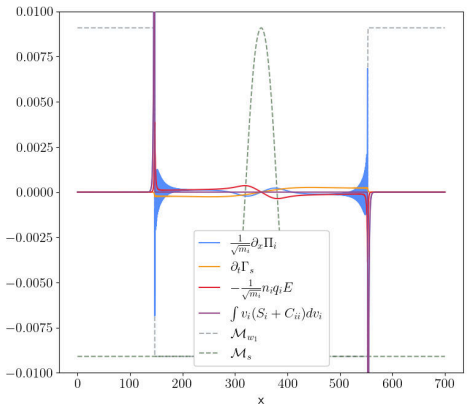
The fluid convergence results indicate that high-order schemes may be more useful for simulating regions whose behaviour is sufficiently described by a fluid model. In these areas the kinetic properties, such as the positivity of the distribution function, are less important, so they can be permitted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

The numerical cost of higher-order non-uniform spline schemes was also studied. Higher-order spline schemes are slower than lower-order spline schemes, and non-uniform schemes are slower than uniform schemes. For cubic splines on GPUs, non-uniform schemes are 30% slower than uniform schemes (see Figure 3.5). The reduction in the number of points when using non-uniform schemes in place of uniform schemes must therefore be at least 30% for the simulation to run in a similar time frame on the same machine. In contrast, on 8 OpenMP threads, non-uniform schemes are 85% slower than uniform schemes (see Figure 3.5). The reduction in points must therefore be at least 85% for the simulation to run in a similar time frame. In the simulations described, an 89% reduction in the number of points was obtained for equivalent results. This means that there would be very limited speed gains when moving to non-uniform points, if these simulations are run on OpenMP. However the parallelisation methods used by GPUs seem more effective, therefore allowing non-uniform simulations to run 5.5 times faster than uniform simulations providing equivalent results. GPU parallelisation is therefore an important tool when accelerating non-uniform schemes based on splines.

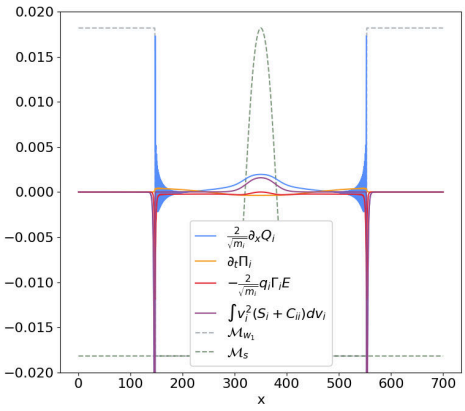
These simulations provide a clear path to the implementation of non-uniform points in the GYSELA code. The memory gains there should be even larger than those seen in this chapter as small reductions in one dimension lead to large reductions in the total number of points in a 5D simulation.



(a) Density conservation, equation (3.33).



(b) Velocity conservation, equation (3.34).



(c) Energy conservation, equation (3.35).

Figure 3.10.: Terms from the conservation equations (3.33)- (3.35) for ions for a simulation with 1500 equidistant points in the spatial dimension, 500 equidistant points in the velocity dimension at time t=2000, and splines of degree 3. The axis of ordinates is truncated to illustrate the shape of the functions.

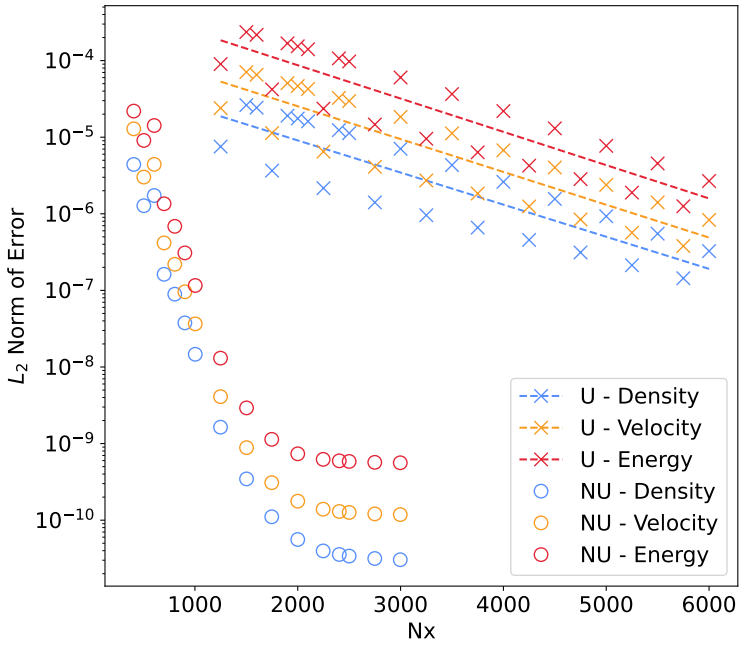


Figure 3.12.: Conservation errors as described in Section 3.2.2 at time t=2000 with 2000 points in the velocity dimension. "U - X" indicates the X conservation for uniform splines of degree 3, while "NU - X" indicates the X conservation for non-uniform splines of degree 3. The saturation of the error at 10^{-9} is expected due to the truncation error discussed in Section A.

4. Local Splines

4.1. Introduction

Semi-Lagrangian advection is a well-known technique which is used for a variety of different simulations including plasma simulations (Cheng and Knorr 1976; Sonnendrücker, Roche, Bertrand, et al. 1999; V. Grandgirard, Abiteboul, J. Bigot, et al. 2016) and weather simulations (Bates and McDonald 1982; Staniforth and Côté 1991). The stability of this method has been proved for linear and quadratic interpolation (Bates and McDonald 1982) as well as for the spline interpolation of arbitrary degree (Besse and Mehrenberger 2008), and the Lagrange interpolation of arbitrary degree (Besse and Mehrenberger 2008). However to the author's knowledge the stability has not been proven for non-uniform spline interpolation.

The simulations which use this method are often massively parallel (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016). While splines are a useful tool for accurately modelling a function, they can be cumbersome in massively parallel simulations, as all data required for the interpolation must be found on the same node. In the case of high-dimensional problems such as plasma physics problems (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016), this therefore requires a large amount of data transfer between nodes. Local splines present themselves as a solution to this problem, by reducing the amount of data transfer required. A previous attempt to provide an effective method for local splines has been made by Nicolas Crouseilles, Guillaume Latu, and Sonnendrücker 2009. Here we present a similar method which also works for non-uniform points, and the proof of the stability of this method.

The proposed method is studied in the context of an advection equation. In Section 4.2 I describe the problem that we would like to solve. In Section 4.2.1 I describe the necessary spline construction for each local splines, such that the stability can be proven. In Section 4.3 the stability of the method is proven. Finally in Section 4.4 I describe how to use these local splines to create a parallel Vlasov-Poisson simulation.

The convergence of semi-Lagrangian schemes on the Vlasov-Poisson system has already been examined by many authors, see for example (Filbet 2001; Besse 2004; Besse 2008; Besse and Mehrenberger 2008; Campos Pinto and Mehrenberger 2008; Bostan and Nicolas Crouseilles 2009; Charles, Després, and Mehrenberger 2013; Einkemmer and Ostermann 2014). This makes it a good test bed to examine this method.

4.2. Constant Semi-Lagrange Advection with Local Spline Interpolation

We consider the following periodic system describing an advection equation:

$$\begin{cases} \partial_t u = v \partial_x u \\ u(0,x) = f(x) \\ u(t,x) = u(t, x + (b-a)) \end{cases} \tag{4.1}$$

where v is a constant representing respectively the velocity of the advection, and $u(t,x)$ and $f(x)$ are $(b-a)$-periodic functions in the periodic Sobolev space $H^k_{per[a,b]}$, the space of functions with $k \geq 0$ weak derivatives in L_2.

This equation is solved using the backward semi-Lagrangian method (Sonnendrücker, Roche, Bertrand, et al. 1999). This method evaluates a function on a grid $\{x_i\}$. The values $u(t_{n+1}, x_i)$ of the function at time t_{n+1} are calculated from the values $u(t_n, x_i)$ at the previous time step t_n by tracing trajectories from the grid points x_i, back to their position at time t_n, and using an approximation of the function $u(t_n, x)$ at this position to provide the updated value. This is possible as the function is constant along the characteristics. These trajectories are known as characteristics and the location at the previous time step is known as the foot of the characteristic $x^\star_{n+1,i}$. In the case of constant advection, as considered here, the semi-Lagrangian method is simple as the feet of the characteristics can be found exactly. The only complexity comes from the approximation of the function $u(t_n, x)$ from the values $u(t_n, x_i)$ in order to obtain the updated values $u(t_{n+1}, x_i) = u(t_n, x^\star_{n+1,i})$. The function $u(t_n, x)$ is approximated using local spline interpolation.

4.2.1. Local Spline Interpolation

I now describe how the proposed local splines interpolate a function $f(x)$ over the domain $[a,b]$. This domain is divided into n_s local splines $S_{i,2m+1}(x)$ of degree $2m+1$ or less, which are each defined on a domain $[a_i, b_i]$ such that:

$$a = a_0 < b_0 = a_1 < b_1 \ldots b_{n_s-2} = a_{n_s-1} < b_{n_s-1} = b$$

Each spline is defined on $n_{c,i}$ cells which are bounded by $n_{c,i}+1$ break points $z_{i,j}$ such that:

$$a_i = z_{i,0} < z_{i,1} < \cdots < z_{i,n_{c,i}} = b_i.$$

For this problem to be well defined, $n_{b,i} = n_{c,i} + 2m + 1$ conditions must be provided, where $n_{b,i}$ is the number of basis functions of the i-th spline. For local splines we use

the following conditions:

$$S_{i,2m+1}(z_{i,j}) = f(z_{i,j}) \qquad \forall 0 \leq j < n_{c,i}+1 \tag{4.2}$$

$$\frac{\partial^k}{\partial x^k} S_{i,2m+1}(a_i) = \frac{\partial^k}{\partial x^k} f(a_i) \qquad \forall 0 \leq k \leq m \tag{4.3}$$

$$\frac{\partial^k}{\partial x^k} S_{i,2m+1}(b_i) = \frac{\partial^k}{\partial x^k} f(b_i) \qquad \forall 0 \leq k \leq m \tag{4.4}$$

In other words, we use interpolation points found at the break points, and Hermite boundary conditions. We will use $\mathcal{I}$ to denote the interpolation operator which constructs a spline respecting Equations (4.2)-(4.4). Figure 4.1 shows the interpolation of $f(x) = \sin(x)$ for cubic splines using $n_s = 3$ local splines and the values necessary to define these splines.

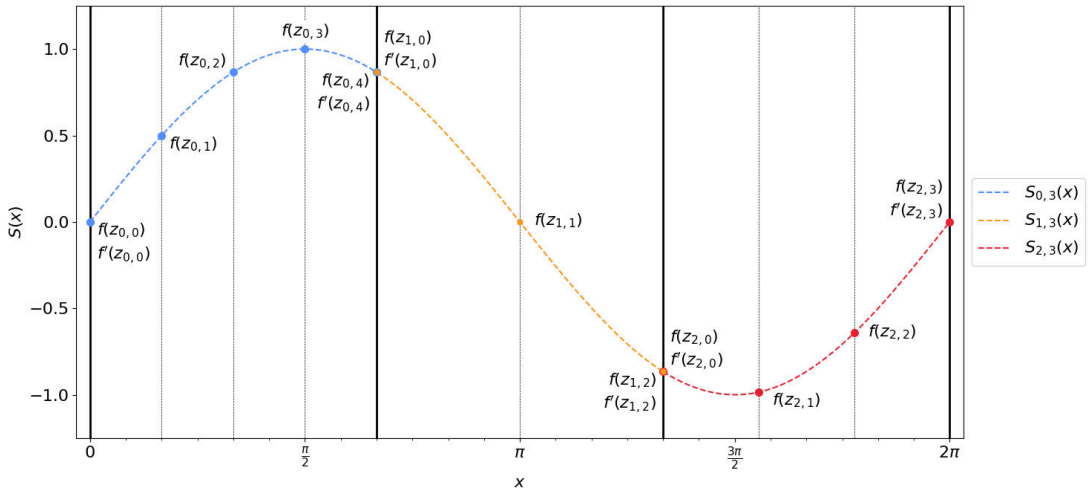


Figure 4.1.: An interpolation of $f(x) = sin(x)$ for cubic splines using $n_s = 3$ local splines, defined using the values and derivatives shown.

The complete spline:

$$S(x) = \begin{cases} S_0(x) & \text{if } x \in [a_0, b_0] \\ \vdots & \\ S_{n_s-1}(x) & \text{if } x \in [a_{n_s-1}, b_{n_s-1}] \end{cases} \tag{4.5}$$

is therefore equivalent to a spline defined on the domain $[a,b]$ with reduced smoothness at the points $a_i = b_{i-1}$, $\forall 1 \leq i < n_s$. This spline $S(x)$ is therefore $\mathcal{C}^m([a,b])$, while each of the local splines $S_i(x)$ is $\mathcal{C}^{2m}([a_i,b_i])$

Let us note $p_n(x)$, the spline approximation of the solution to Equation (4.1) at time $t_n = n\tau$ using the semi-Lagrangian method on the local splines described above, where τ is the time step of the method. At time $t = 0$, $p_0(x)$ is therefore simply the

spline interpolation of the initial function $u(0,x) = f(x)$:

$$p_0(x) = \mathcal{I}f(x) \tag{4.6}$$

We use $\mathcal{A}$ to denote the exact advection operator,

$$\mathcal{A}p_n(x) = \begin{cases} p_n(x - v\tau) & \text{if } x - v\tau \in [a,b] \\ p_n(x + b - a - v\tau) & \text{otherwise} \end{cases} \tag{4.7}$$

The periodicity of the system is used to ensure that $x - v\tau \in [a,b]$. As $p_{n+1}(x)$ is constructed from $p_n(x)$ using the semi-Lagrangian method with constant advection, the proposed numerical scheme is therefore summarised as follows:

$$p_0(x) = \mathcal{I}f(x) \tag{4.8}$$
$$p_{n+1}(x) = \mathcal{I}\mathcal{A}p_n(x) \tag{4.9}$$

Written more explicitly, the equations that the interpolator $\mathcal{I}$ must satisfy are:

$$p_0(z_{i,j}) = f(z_{i,j}) \qquad \forall 0 \le i < n_s, \quad \forall 0 \le j \le n_c \tag{4.10}$$
$$\frac{\partial^k}{\partial x^k} p_0(a_0) = \frac{\partial^k}{\partial x^k} f(a_0) \qquad \forall 0 \le k \le m \tag{4.11}$$
$$\frac{\partial^k}{\partial x^k} p_0(b_i) = \frac{\partial^k}{\partial x^k} f(b_i) \qquad \forall 0 \le k \le m, \quad \forall 0 \le i < n_s \tag{4.12}$$
$$p_{n+1}(z_{i,j}) = p_n(z_{i,j} - v\tau) \qquad \forall 0 \le i < n_s, \quad \forall 0 \le j \le n_c \tag{4.13}$$
$$\frac{\partial^k}{\partial x^k} p_{n+1}(a_i) = \frac{\partial^k}{\partial x^k} p_n(a_i - v\tau) \qquad \forall 0 \le k \le m, \quad \forall 0 \le i < n_s \tag{4.14}$$
$$\frac{\partial^k}{\partial x^k} p_{n+1}(b_i) = \frac{\partial^k}{\partial x^k} p_n(b_i - v\tau) \qquad \forall 0 \le k \le m, \quad \forall 0 \le i < n_s \tag{4.15}$$

4.3. Stability of Semi-Lagrange Advection on Local Splines

I will now prove the stability of the backward semi-Lagrangian advection on the local splines. To do this, I first define an inner-product and its associated norm:

$$\langle f, g \rangle = \int_a^b f(x) g(x) dx \qquad \|f\|^2 = \langle f, f \rangle \tag{4.16}$$

Note that the advection operator preserves this norm thanks to the periodicity of the function:

$$\begin{aligned}\left\|\mathcal{A}f\right\|^2 &= \int_a^b \mathcal{A}f(x)\mathcal{A}f(x)dx \\ &= \int_{a+v\tau}^{b} f(x-v\tau)f(x-v\tau)dx + \int_a^{a+v\tau} f(x+b-a-v\tau)f(x+b-a-v\tau)dx \\ &= \int_a^{b-v\tau} f(y)f(y)dy + \int_{b-v\tau}^{b} f(y)f(y)dz = \int_a^b f(y)f(y)dy = \left\|f\right\|^2 \end{aligned} \tag{4.17}$$

For this proof I will also use the periodic Sobolev spaces $H^k_{per[a,b]}$, the space of functions with $k \geq 0$ weak derivatives in L_2. In other words a function f is in $H^k_{per[a,b]}$ if:

$$\int_a^b |D^k f(x)|^2 dx < \infty \tag{4.18}$$

where $D^k f(x)$ is the k-th weak derivative of the function $f(x)$. I note that this means that $H^k_{per[a,b]} \subset H^r_{per[a,b]}$ for all $k \geq r$.

Lemma 1. *The result of a projection of a function $f \in H^{m+1}_{per[a,b]}$ onto the spline space with the local spline interpolation is such that:*

$$\mathcal{I}f \in H^{m+1}_{per[a,b]} \tag{4.19}$$

Proof. Each local spline is a piecewise polynomial of degree $2m+1$ or less, and the global spline is $\mathcal{C}^m$. We can prove that the $m+1$-th derivative of a piecewise polynomial of degree $2m+1$ is a piecewise polynomial function of degree m:

$$\int_a^b \mathcal{I}f(x)\frac{\partial^{m+1}\phi(x)}{\partial x^{m+1}}dx = \sum_{i=0}^{n_s-1}\sum_{j=0}^{n_{c,i}-1}\int_{z_{i,j}}^{z_{i,j+1}} \mathcal{I}f(x)\frac{\partial\phi(x)}{\partial x}dx \tag{4.20}$$

$$= \sum_{i=0}^{n_s-1}\sum_{j=0}^{n_{c,i}-1} \frac{\partial^m\phi(x)}{\partial x^m}\mathcal{I}f(x)|_{z_{i,j}}^{z_{i,j+1}} - \int_{z_{i,j}}^{z_{i,j+1}} \frac{\partial}{\partial x}\mathcal{I}f(x)\frac{\partial^m\phi(x)}{\partial x^m}dx \tag{4.21}$$

$$\begin{aligned}&= \sum_{k=0}^{m}\sum_{i=0}^{n_s-1} \frac{\partial^k\phi(x)}{\partial x^k}D^{m-k}\mathcal{I}f(x)|_{z_{i,j}}^{z_{i,j+1}} \\ &\quad + (-1)^{m+1}\int_{z_{i,j}}^{z_{i,j+1}} D^{m+1}\mathcal{I}f(x)\phi(x)dx\end{aligned} \tag{4.22}$$

where $D^{m-k}\mathcal{I}f(x)$ is well defined on the interval $[a_i, b_i]$ for $k \leq m$ as it is a piecewise polynomial. As the global spline $\mathcal{I}f(x) \in \mathcal{C}^m$ we have:

$$\int_a^b \mathcal{I}f(x)\frac{\partial^{m+1}\phi(x)}{\partial x^{m+1}}dx = \frac{\partial^k\phi(x)}{\partial x^k}D^{m-k}\mathcal{I}f(x)\Big|_a^b + (-1)^{m+1}\int_a^b D^{m+1}\mathcal{I}f(x)\phi(x)dx \tag{4.23}$$

Furthermore, using the periodicity of the function we have:

$$\int_a^b \mathcal{I}f(x)\frac{\partial^{m+1}\phi(x)}{\partial x^{m+1}}dx = (-1)^{m+1}\int_a^b D^{m+1}\mathcal{I}f(x)\phi(x)dx \tag{4.24}$$

□

Theorem 2. *For all* $f, g \in H^{m+1}_{per[a,b]}$,

$$\left\langle D^{m+1}\mathcal{I}f, D^{m+1}(g-\mathcal{I}g)\right\rangle = 0$$

Proof. Integrating by parts we have:

$$\begin{aligned}
&\left\langle D^{m+1}\mathcal{I}f, D^{m+1}(g-\mathcal{I}g)\right\rangle \\
&= \sum_{i=0}^{n_s-1}\sum_{j=0}^{n_{c,i}-1}\int_{z_{i,j}}^{z_{i,j+1}} D^{m+1}\mathcal{I}f(x)D^{m+1}(g-\mathcal{I}g)(x)dx && (4.25)\\
&= \sum_{i=0}^{n_s-1}\sum_{j=0}^{n_{c,i}-1}\left[\sum_{k=0}^{m}(-1)^k D^{m+1+k}\mathcal{I}f(x)D^{m-k}(g-\mathcal{I}g)(x)\Big|_{z_{i,j}}^{z_{i,j+1}}\right. \\
&\quad \left.+(-1)^{m+1}\int_{z_{i,j}}^{z_{i,j+1}} D^{2m+2}\mathcal{I}f(x)(g-\mathcal{I}g)(x)dx\right] && (4.26)\\
&= \sum_{i=0}^{n_s-1}\sum_{j=0}^{n_{c,i}-1}\left[\sum_{k=0}^{m}(-1)^{m-k} D^{2m+1-k}\mathcal{I}f(x)D^{k}(g-\mathcal{I}g)(x)\Big|_{z_{i,j}}^{z_{i,j+1}}\right. \\
&\quad \left.+(-1)^{m+1}\int_{z_{i,j}}^{z_{i,j+1}} D^{2m+2}\mathcal{I}f(x)(g-\mathcal{I}g)(x)dx\right] && (4.27)
\end{aligned}$$

Since $\mathcal{I}f(x)$ on the interval $[z_{i,j}, z_{i,j+1}]$ is a polynomial of degree no greater than $2m+1$:

$$D^{2m+2}\mathcal{I}f(x) = 0 \qquad\qquad \forall x \in [z_{i,j}, z_{i,j+1}]$$

This leaves:

$$\begin{aligned}
&\left\langle D^{m+1}\mathcal{I}f, D^{m+1}(g-\mathcal{I}g)\right\rangle \\
&= \sum_{i=0}^{n_s-1}\sum_{j=0}^{n_{c,i}-1}\sum_{k=0}^{m}(-1)^{m-k} D^{2m+1-k}\mathcal{I}f(x)D^{k}(g-\mathcal{I}g)(x)\Big|_{z_{i,j}}^{z_{i,j+1}} \\
&= \sum_{i=0}^{n_s-1}\sum_{k=0}^{m}\left[(-1)^{m-k} D^{2m+1-k}\mathcal{I}f(b_i)D^{k}(g-\mathcal{I}g)(b_i)\right. \\
&\quad -(-1)^{m-k} D^{2m+1-k}\mathcal{I}f(a_i)D^{k}(g-\mathcal{I}g)(a_i) \\
&\quad \left.-\sum_{j=1}^{n_{c,i}-1}(-1)^{m-k} D^{2m+1-k}\mathcal{I}f(x)D^{k}(g-\mathcal{I}g)\Big|_{z_{i,j-}}^{z_{i,j+}}\right] && (4.28)
\end{aligned}$$

where $f(z_{i,j-})$ denotes the left limit of $f(x)$ at $z_{i,j}$, and $z_{i,j+}$ denotes the right limit of

$f(x)$ at $z_{i,j}$. As the interpolating points are at the knots:

$$\mathcal{I}g(z_{i,j}) = g(z_{i,j}) \qquad \forall 0 \le i \le n_s, \quad \forall 0 \le j \le n_{c,i}$$

Furthermore, thanks to the continuity of both g and the spline interpolations:

$$D^k(g-\mathcal{I}g)(z_{i,j-}) = D^k(g-\mathcal{I}g)(z_{i,j+}), \qquad 0 \le k \le 2m \tag{4.29}$$

$$D^k\mathcal{I}f(x_-) = D^k\mathcal{I}f(x_+), \qquad 0 \le k \le 2m \tag{4.30}$$

This leaves:

$$\begin{aligned}\left\langle D^{m+1}\mathcal{I}f, D^{m+1}(g-\mathcal{I}g)\right\rangle = \sum_{i=0}^{n_s-1}\sum_{k=1}^{m}\Big[&(-1)^{m-k}D^{2m+1-k}\mathcal{I}f(b_i D^k(g-\mathcal{I}g)(b_i) \\ &- (-1)^{m-k}D^{2m+1-k}\mathcal{I}f(a_i)D^k(g-\mathcal{I}g)(a_i)\Big]\end{aligned} \tag{4.31}$$

The local splines apply Hermite boundary conditions between splines (i.e. at the knots k_j). Hermite boundary conditions involve specifying the first m derivatives at the boundary. This means that the interpolation error for these derivatives is also zero:

$$D^k(g-\mathcal{I}g)(a_j) = D^k(g-\mathcal{I}g)(b_j) = 0 \qquad 0 \le k \le m \quad 0 \le j < n_s$$

□

Corollary 1. *Let $f(x) \in H^{m+1}_{per[a,b]}$, let $\mathcal{I}$ be the interpolation operator previously defined, then*

$$\left\|D^{m+1}(f-\mathcal{I}f)\right\| \le \left\|D^{m+1}f\right\| \tag{4.32}$$

Proof. We have:

$$\left\|D^{m+1}f\right\|^2 = \left\|D^{m+1}(f(x)-\mathcal{I}f(x)) - D^{m+1}\mathcal{I}f(x)\right\|^2 \tag{4.33}$$

$$\begin{aligned}&= \left\|D^{m+1}(f(x)-\mathcal{I}f(x))\right\|^2 + \left\|D^{m+1}\mathcal{I}f(x)\right\|^2 \\ &\quad + 2\langle D^{m+1}\mathcal{I}f(x), D^{m+1}(f(x)-\mathcal{I}f(x))\rangle\end{aligned} \tag{4.34}$$

The final term is zero, thanks to Theorem 2. □

Lemma 2. *Let $f \in H^{m+1}_{per[a,b]}$, let $\{z_{0,0},\dots,z_{n_s-1,n_{c,i}}\}$ be the mesh upon which the interpolation operator $\mathcal{I}$ is defined, and let $h = \max_{i<n_s}\left(\max_{j\le n_{c,i}}(z_{i,j+1}-z_{i,j})\right)$. There exists a constant C dependent upon k and r such that:*

$$\left\|D^r(f-\mathcal{I}f)\right\| \le Ch^{m+1-r}\left\|D^{m+1}(f-\mathcal{I}f)\right\| \le Ch^{m+1-r}\left\|D^{m+1}f\right\| \quad \forall 0 \le r \le m+1 \tag{4.35}$$

We note that in the above statements and in what follows, C stands for any constant independent of h.

Proof. The proof of this expression is derived from Theorem 7 in (Schultz and Varga 1967), but I summarise it here for completeness.

I first note that the simplest case $r = m+1$ is equivalent to Corollary 1. I will now show the proof for $r < m+1$.

The interpolation operator $\mathcal{I}$ satisfies equations (4.2) - (4.3). Therefore on each local spline there are $n_{c,i}+1$ points $\xi_{i,j,0}$ where:

$$f(\xi_{i,j,0}) - \mathcal{I}f(\xi_{i,j,0}) = 0 \qquad \forall 0 \le i < n_s, \quad \forall 0 \le j \le n_{c,i} \tag{4.36}$$

Using Rolle's theorem we see that there are $n_{c,i}$ points $\xi_{i,j,1}$ where:

$$D^1 f(\xi_{i,j,1}) - D^1 \mathcal{I}f(\xi_{i,j,1}) = 0 \qquad \forall 0 \le i < n_s, \quad \forall 1 \le j < n_{c,i}+1 \tag{4.37}$$

In addition the extremal points a_i and b_i where the Hermite boundary conditions are applied also respect this condition which allows us to define $\xi_{i,0,1} = a_i$ and $\xi_{i,n_{c,i}+1,q} = b_i$. We apply this criteria recursively to obtain $n_{\xi,r} = n_{c,i}+1+r$ points such that:

$$D^r f(\xi_{i,j,r}) - D^r \mathcal{I}f(\xi_{i,j,r}) = 0 \qquad \forall 0 \le i < n_s, \quad \forall 0 \le j < n_{\xi,r} \tag{4.38}$$

with $r \le m$. with $\xi_0 = a_i$ and $\xi_{n_\xi - 1} = b_i$. We therefore have:

$$\left\| D^r(f - \mathcal{I}f) \right\|^2 = \int_a^b (D^r(f(x) - \mathcal{I}f(x)))^2 dx = \sum_{i=0}^{n_s-1} \sum_{j=0}^{n_\xi - 2} \int_{\xi_{i,j,r}}^{\xi_{i,j+1,r}} (D^r(f(x) - \mathcal{I}f))^2 dx \tag{4.39}$$

Using the Poincaré-Wirtinger inequality we then obtain:

$$\left\| D^r(f - \mathcal{I}f) \right\|^2 \le \left(\frac{(r+1)h}{\pi} \right)^2 \sum_{i=0}^{n_s-1} \sum_{j=0}^{n_\xi - 2} \int_{\xi_{i,j,r}}^{\xi_{i,j+1,r}} (D^{r+1}(f(x) - \mathcal{I}f))^2 dx \tag{4.40}$$

as $\xi_{i,j+1,r} - \xi_{i,j,r} \le (r+1)h$. The proof of equation (4.35) is therefore constructed recursively from the following two identities:

$$\left\| D^r(f - \mathcal{I}f) \right\|^2 \le \left(\frac{(r+1)h}{\pi} \right)^2 \left\| D^{r+1}(f - \mathcal{I}f) \right\|^2 \tag{4.41}$$

$$\left\| D^m(f - \mathcal{I}f) \right\|^2 \le \left(\frac{(m+1)h}{\pi} \right)^2 \left\| D^{m+1}(f - \mathcal{I}f) \right\|^2 \tag{4.42}$$

□

Lemma 3. *Let $f \in H^{2m+2}_{per[a,b]}$, let $\{z_{0,0}, \ldots, z_{n_s-1,n_{c,i}}\}$ be the mesh upon which the interpolation operator $\mathcal{I}$ is defined, and let $h = \max_{i<n_s}\left(\max_{j\le n_{c,i}}(z_{i,j+1} - z_{i,j})\right)$. There exists a constant C dependent upon k and r such that:*

$$\left\| D^r(f - \mathcal{I}f) \right\| \le C h^{2m+2-r} \left\| D^{2m+2} f \right\| \qquad \forall 0 \le r \le m+1 \tag{4.43}$$

Proof. The proof of this expression is derived from Theorem 9 in (Schultz and Varga

1967), but I summarise it here for completeness.

Using theorem 2 we have:

$$\left\|D^{m+1}f\right\|^2 = \left\|D^{m+1}(f-\mathcal{I}f)\right\|^2 + \left\|D^{m+1}\mathcal{I}f\right\|^2 \tag{4.44}$$

Let us apply Cauchy-Schwarz's theorem and integration by parts as in equation (4.28):

$$\left\|D^{m+1}(f-\mathcal{I}f)\right\|^2 \leq \left\|D^{2m+2}(f-\mathcal{I}f)\right\| \left\|f-\mathcal{I}f\right\| \tag{4.45}$$

Remembering that $H^{2m+2}_{per[a,b]} \subset H^{m+1}_{per[a,b]}$ we can use Lemma 2:

$$\left\|D^{m+1}(f-\mathcal{I}f)\right\|^2 \leq Ch^{m+1}\left\|D^{2m+2}(f-\mathcal{I}f)\right\| \left\|D^{m+1}(f-\mathcal{I}f)\right\| \tag{4.46}$$

by cancelling the positive term $\left\|D^{m+1}(f-\mathcal{I}f)\right\|$ we have an equation which we can use to complete the proof:

$$\begin{aligned} \left\|D^{r}(f-\mathcal{I}f)\right\| &\leq Ch^{m+1-r}\left\|D^{m+1}(f-\mathcal{I}f)\right\| && (4.47)\\ &\leq Ch^{2m+2-r}\left\|D^{2m+2}f\right\| && (4.48) \end{aligned}$$

□

I now summarise the results from Lemmas 2 and 3 that I will use in this proof. They are that the local spline interpolation operator $\mathcal{I}$ satisfies:

$$\begin{aligned} \left\|g-\mathcal{I}g\right\| &\leq Ch^{2m+2}\left\|D^{2m+2}g\right\| && \text{for } g \in H^{2m+2}_{per[a,b]} && (4.49)\\ \left\|D^{m+1}\left(g-\mathcal{I}g\right)\right\| &\leq Ch^{m+1}\left\|D^{2m+2}g\right\| && \text{for } g \in H^{2m+2}_{per[a,b]} && (4.50)\\ \left\|g-\mathcal{I}g\right\| &\leq Ch^{m+1}\left\|D^{m+1}g\right\| && \text{for } g \in H^{m+1}_{per[a,b]} && (4.51) \end{aligned}$$

The method used to prove stability closely follows the method used in (Goodrich, Hagstrom, and Lorenz 2006). Like them we note that if we set $g = q - \mathcal{I}q$ in equation (4.51) and observe that $\mathcal{I}(q-\mathcal{I}q) = 0$ then we obtain:

Corollary 2.

$$\left\|q-\mathcal{I}q\right\| \leq Ch^{m+1}\left\|D^{m+1}(q-\mathcal{I}q)\right\| \qquad \text{for } q \in H^{m+1}_{per[a,b]} \tag{4.52}$$

Let $u_n \in H^{m+1}_{per[a,b]}$ be the solution to the equation (4.1):

$$u_n = u(n\tau,\cdot) \tag{4.53}$$

The error $e_n \in H^{m+1}_{per[a,b]}$ is therefore defined as:

$$e_n = u_n - p_n \tag{4.54}$$

Substituting equation (4.53) into equation (4.9) allows us to define a local truncation

error η_n:

$$u_{n+1} = \mathcal{I}\mathcal{A}u_n + \eta_n \tag{4.55}$$
$$\eta_n = u_{n+1} - \mathcal{I}u_{n+1} \tag{4.56}$$

where we have used the fact that the advection operator is exact. We can therefore see that the truncation error is an interpolation error.

Theorem 3. *Let $\lambda = \tau/h > 0$ and let $T > 0$ be fixed. Suppose $f \in H^{2m+2}_{per[a,b]}$ a periodic function $f(x) = f(x+(b-a))$ and $p_n = \mathcal{I}f$. Then there exists a constant C, independent of h such that:*

$$\|u_n - p_n\| \leq Ch^{2m+1} \|D^{2m+2} f\| \quad \text{for } 0 \leq n\tau \leq T$$

Proof. Combining equations (4.55) and (4.9) we obtain another expression for the error:

$$e_{n+1} = \mathcal{I}\mathcal{A}u_n + \eta_n - \mathcal{I}\mathcal{A}p_n = \mathcal{I}\mathcal{A}e_n + \eta_n \tag{4.57}$$

Using equation (4.49) and the definition of the truncation error from equation (4.56) we have:

$$\|\eta_n\| \leq Ch^{2m+2} \|D^{2m+2} u_{n+1}\| \tag{4.58}$$

We note that $u_{n+1} = \mathcal{A}u_n$ and $\mathcal{A}$ preserves the norm so $\|D^{2m+2}u_{n+1}\| = \|D^{2m+2}f\|$

I now examine the L_2-norm of e_{n+1}. Using the triangle inequality we have:

$$\begin{aligned}
\|e_{n+1}\| &\leq \|\mathcal{I}\mathcal{A}e_n\| + \|\eta_n\| \\
&\leq \|\mathcal{I}\mathcal{A}e_n\| + Ch^{2m+2} \|D^{2m+2} f\| \\
&= \|\mathcal{A}e_n - (\mathcal{A}e_n - \mathcal{I}\mathcal{A}e_n)\| + Ch^{2m+2} \|D^{2m+2} f\| \\
&\leq \|\mathcal{A}e_n\| + \|\mathcal{A}e_n - \mathcal{I}\mathcal{A}e_n\| + Ch^{2m+2} \|D^{2m+2} f\|
\end{aligned} \tag{4.59}$$

Using the fact that the advection operator preserves the norm we can write this more succinctly as:

$$\|e_{n+1}\| \leq \|e_n\| + \|\mathcal{A}e_n - \mathcal{I}\mathcal{A}e_n\| + Ch^{2m+2} \|D^{2m+2} f\| \tag{4.60}$$

Using Corollary 2 we obtain:

$$\|\mathcal{A}e_n - \mathcal{I}\mathcal{A}e_n\| \leq Ch^{m+1} \|D^{m+1}(\mathcal{A}e_n - \mathcal{I}\mathcal{A}e_n)\| \tag{4.61}$$

There is no obvious bound for the right hand side of this equation. We will therefore work with similar equations until a bound can be found. Firstly we examine $\|D^{m+1}e_{n+1}\|$ using equation (4.55) and theorem 2:

$$\|D^{m+1}e_{n+1}\| = \|D^{m+1}\mathcal{I}\mathcal{A}e_n + D^{m+1}\eta_n\| \tag{4.62}$$
$$\|D^{m+1}e_{n+1}\|^2 = \|D^{m+1}\mathcal{I}\mathcal{A}e_n\|^2 + \|D^{m+1}\eta_n\|^2 \tag{4.63}$$

Using equations (4.50) and (4.56) we therefore have:

$$\left\|D^{m+1}e_{n+1}\right\|^2 \leq \left\|D^{m+1}\mathcal{I}\mathcal{A}e_n\right\|^2 + \left(Ch^{m+1}\left\|D^{2m+2}f\right\|\right)^2 \tag{4.64}$$

We now consider the term $D^{m+1}\mathcal{A}e_n$:

$$\left\|D^{m+1}\mathcal{A}e_n\right\|^2 = \left\|D^{m+1}\mathcal{I}\mathcal{A}e_n + D^{m+1}(\mathcal{A}e_n - \mathcal{I}\mathcal{A}en)\right\|^2 \tag{4.65}$$

$$= \left\|D^{m+1}\mathcal{I}\mathcal{A}e_n\right\|^2 + \left\|D^{m+1}(\mathcal{A}e_n - \mathcal{I}\mathcal{A}e_n)\right\|^2 \tag{4.66}$$

which is once more thanks to corollary 2.

In order to simplify calculations let us now define 2 terms:

$$\epsilon_n := \left\|D^{m+1}e_n\right\| \tag{4.67}$$

$$\delta_n := \left\|D^{m+1}(\mathcal{A}e_n - \mathcal{I}\mathcal{A}e_n)\right\| \tag{4.68}$$

Inserting equations (4.67) and (4.68) into equations (4.64) and (4.66), and using the fact that $\left\|D^{m+1}\mathcal{A}e_n\right\| = \left\|D^{m+1}e_n\right\|$, we obtain:

$$\epsilon_{n+1}^2 \leq \epsilon_n^2 - \delta_n^2 + Ch^{2m+2}\left\|D^{2m+2}f\right\|^2 \tag{4.69}$$

$$\leq \epsilon_n^2 + Ch^{2m+2}\left\|D^{2m+2}f\right\|^2 \tag{4.70}$$

$$\epsilon_n^2 \leq \epsilon_0^2 + nCh^{2m+2}\left\|D^{2m+2f}\right\|^2 \tag{4.71}$$

$$\delta_n^2 \leq \epsilon_n^2 - \epsilon_{n+1}^2 + Ch^{2m+2}\left\|D^{2m+2}f\right\|^2 \tag{4.72}$$

This provides us with the missing boundary for equation (4.60):

$$\|e_{n+1}\| \leq \|e_n\| + Ch^{m+1}\delta_n + Ch^{2m+2}\left\|D^{2m+2}f\right\| \tag{4.73}$$

$$\|e_n\| \leq \|e_0\| + Ch^{m+1}\sum_{k=0}^{n-1}\delta_k + nCh^{2m+2}\left\|D^{2m+2}f\right\| \tag{4.74}$$

Using the Cauchy-Schwarz inequality we have:

$$\sum_{k=0}^{n-1}\delta_k \leq \left[\sum_{k=0}^{n-1}1\sum_{k=0}^{n-1}\delta_k^2\right]^{\frac{1}{2}} \tag{4.75}$$

$$\sum_{k=0}^{n-1}\delta_k^2 \leq \sum_{k=0}^{n-1}\left[\epsilon_k^2 - \epsilon_{k+1}^2 + Ch^{2m+2}\left\|D^{2m+2}f\right\|^2\right] \tag{4.76}$$

$$= \epsilon_0^2 - \epsilon_n^2 + nCh^{2m+2}\left\|D^{2m+2}f\right\|^2 \tag{4.77}$$

$$\leq \epsilon_0^2 + nCh^{2m+2}\left\|D^{2m+2}f\right\|^2 \tag{4.78}$$

$$= \left\|D^{m+1}u_0 - \mathcal{I}u_0\right\|^2 + nCh^{2m+2}\left\|D^{2m+2}f\right\|^2 \tag{4.79}$$

$$\leq \left[Ch^{m+1}\left\|D^{2m+2}f\right\|\right]^2 + nCh^{2m+2}\left\|D^{2m+2}f\right\|^2 \tag{4.80}$$

We note that $0 \leq n\tau \leq T$ and $\lambda = \tau / h >$, therefore:

$$n \leq \frac{T}{\lambda h} \tag{4.81}$$

Therefore we can simplify equation (4.75) further to:

$$\left(\sum_{k=0}^{n-1} \delta_k\right)^2 \leq C n^2 h^{2m+2} \left\| D^{2m+2} f \right\|^2 \tag{4.82}$$

$$\leq \frac{T^2}{\lambda^2} C n^2 h^{2m} \left\| D^{2m+2} f \right\|^2 \tag{4.83}$$

$$= C h^{2m} \left\| D^{2m+2} f \right\|^2 \tag{4.84}$$

Finally we insert this into equation (4.74) to obtain the final error bound:

$$\|e_n\| \leq \|u_0 - \mathcal{I} u_0\| + C h^{2m+1} \left\| D^{2m+2} f \right\| + C h^{2m+1} \left\| D^{2m+2} f \right\| \tag{4.85}$$

$$\leq C h^{2m+1} \left\| D^{2m+2} f \right\| \tag{4.86}$$

□

4.3.1. Extension to Semi-Lagrangian Advection on Non-Periodic Splines

Having shown the stability for the periodic case on local splines, we can use this information to deduce boundary conditions for the non-periodic case which will give rise to a stable simulation.

For a spline of degree $2m+1$, an acceptable boundary condition would be one where the function outside the bounds is described by a polynomial of degree at most $2m+1$ whose derivatives match those provided to the Hermite boundary conditions of the spline. In this way the boundary conditions can be exactly described by a spline which means that the above proof can be expanded for a domain going from negative to positive infinity.

One simple example of boundary conditions satisfying these conditions are constant boundary conditions where the value of the function does not vary outside the domain and the Hermite boundary conditions are:

$$D^k p(a_0) = 0 \qquad \forall 0 < k \leq m \tag{4.87}$$

$$D^k p(b_{n_s-1}) = 0 \qquad \forall 0 < k \leq m \tag{4.88}$$

4.4. Parallelisation

In this section I will use the example of a 1D-1V Vlasov-Poisson simulation to show the implementation details of a system using local splines to increase parallelism. This

system describes the evolution of the distribution functions $f_s(t,x,v)$ of a ions ($s=i$) and electrons ($s=e$), in a plasma, on a domain $[x_0,x_N]\times[v_0,v_N]$. It comprises the Vlasov advection equation (4.89) and the Poisson equation (4.90):

$$\left[\partial_t + \frac{1}{\sqrt{m_s}}\left(v\partial_x - q_s\partial_x\phi(t,x)\partial_v\right)\right] f_s(t,x,v) = 0 \tag{4.89}$$

$$\partial_x^2\phi(t,x) = -\rho_q(t,x) \tag{4.90}$$

where m_s and q_s are the mass and charge of the species s, $\phi(t,x)$ is the electric potential, and $\rho_q(t,x)$ is the charge density, defined as:

$$\rho_q(t,x) = \sum_{s\in\{i,e\}} q_s \int_{v_0}^{v_N} f(t,x,v)\,dv \tag{4.91}$$

Equation (4.89) will be solved using Strang splitting (Strang 1968). This breaks the equation into the following two equations:

$$\partial_t f_s(t,x,v) + \frac{v}{m_s}\partial_x f_s(t,x,v) = 0 \tag{4.92}$$

$$\partial_t f_s(t,x,v) - \frac{q_s}{m_s}\partial_x\phi(t,x)\partial_v f_s(t,x,v) = 0 \tag{4.93}$$

I use a predictor-corrector method to solve this system of equations.

Local splines lend themselves to parallelisation using MPI. Each MPI process will describe a sub-domain using a local spline. The local splines can be used in either the spatial or the velocity direction, leaving global splines to describe the remaining direction in the habitual way. Although they could be used in both directions, this would result in a large number of patches, incurring significant communication costs. It is unlikely that the problem described would require a numbers of grid points sufficiently large to offset these costs in both the spatial and velocity direction. In this work we will divide the domain in the spatial dimension only.

Bourne, Munschy, Virginie Grandgirard, et al. 2022 have previously studied a similar system with additional terms in the right hand side of equation (4.89) to model a wall. The system was used for sheath simulations where a large domain is required in the spatial direction. Furthermore, Bourne, Munschy, Virginie Grandgirard, et al. 2022 showed that in the case of sheath simulations, splines which are uniform by zone can be used to decrease memory consumption while retaining a high enough resolution to successfully model the steep gradients that arise near the wall. In the case of global splines, this implies the use of costly non-uniform methods, however the use of local splines in the spatial direction could avoid this constraint by using different spatial discretisations for each local spline.

In Section 4.4.1, I will explain how equation (4.92) can be solved using local splines. In Section 4.4.2, I will explain how equation (4.90) can be solved using local splines.

4.4.1. Advection

The backward semi-Lagrangian method combines the benefits of the well-known PIC and Eulerian methods (Sonnendrücker, Roche, Bertrand, et al. 1999). As previously described in Section 4.2, this method evaluates a function on a grid $\{x_i\}$. The values of the function at time t_{n+1} are calculated from the values at the previous time step t_n by tracing trajectories from the grid points x_i, back to their position at time t_n, and using an approximation of the function at this position to provide the updated value. The approximation is provided by the local splines. The trajectories are known as characteristics and the location at the previous time step is known as the foot of the characteristic $x^\star_{n+1,i}$.

The foot of the characteristic is not necessarily found on the same local spline. It is therefore important to consider how to handle the junction between local splines. In the case of constant advection, such as equation (4.92), the feet of the characteristics are known ahead of time. One option is therefore for each local spline to calculate the values at the feet of the characteristics which are found on its support. They can then share those values with the threads which need those values.

However, in most cases, such as equation (4.93), the feet are not found in the same places throughout the duration of the simulation. An alternative method, using ghost cells, is therefore better adapted to most problems. This introduces a CFL condition to the problem, as the feet of the characteristics must be located inside the local spline domain or its ghost regions. The number of cells n_g in the ghost region must be chosen to ensure that the CFL condition is sufficiently large, while avoiding excessive communication between processes. Figure 4.2 illustrates the ghost cells for a local spline $S_i(x)$. We can see that the size of the cells in the adjacent local splines also affects the CFL condition.

Each ghost region is represented with an additional local spline. If the knots are chosen to coincide with the relevant knots of the adjacent local spline, and the interpolation values, including the Hermite boundary conditions, are provided by the adjacent local spline representation, then, thanks to the unicity of solutions for the spline interpolation problem, the ghost spline will represent the same function as the adjacent local spline. We note that only the n_{bc} boundary conditions at the outer edges need to be provided, as the boundary conditions at the intersection of the ghost splines and the local spline are equal to the boundary conditions of the local spline. Figure 4.2 shows the data which needs to be transferred between processes for a configuration with two ghost cells and splines of degree 3.

The advection scheme in the spatial direction is therefore summarised as follows:

- Construct the local spline using the stored values at the interpolation points and the stored derivatives at the boundary
- Pass $2(n_g + n_{bc})$ values describing the two ghost regions (shown in orange in Figure 4.2) to the neighbouring processes
- Use the backward semi-Lagrangian method to update the values at the break points

- Use the backward semi-Lagrangian method to update the derivatives at the domain boundaries

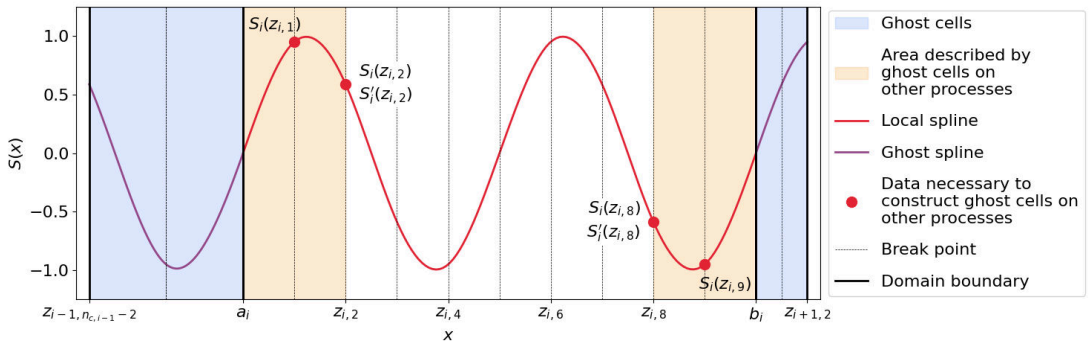


Figure 4.2.: Illustration of the ghost cells and splines for a local spline $S_i(x)$ of degree 3, and the data necessary to construct the equivalent ghost cells for the adjacent local splines.

As we do not distribute the data in the velocity direction, no special treatment of the backward semi-Lagrangian method is required in this case. However, in order to be able to construct the spline in the spatial direction at the next iteration, it is important to advect, not only the distribution function $f(x,v)$, but also its spatial derivative at the boundaries, $\partial_x f(a_i,v)$ and $\partial_x f(b_i,v)$.

4.4.2. Poisson Solver

The Poisson equation (4.90) can be solved using a finite element method. The weak form of equation (4.90) is:

$$\partial_x\phi(t,x)b_{i,d}(x)\Big|_a^b - \int_a^b \partial_x\phi(t,x)\partial_x b_{i,d}(x)dx = -\int_a^b \rho_q(t,x)b_{i,d}(x)dx \quad \forall 0 \le i < n_b \tag{4.94}$$

where $\{b_{i,d}(x)\}$ are the basis functions of splines of degree d, and n_b is the number of basis functions. We impose periodic boundary conditions and break the equation into the sub domains:

$$\sum_{i=0}^{n_s-1}\int_{a_i}^{b_i} \partial_x\phi(t,x)\partial_x b_{i,d}(x)dx = \sum_{i=0}^{n_s-1}\int_{a_i}^{b_i} \rho_q(t,x)b_{i,d}(x)dx \tag{4.95}$$

Each equation of the form:

$$\int_{a_i}^{b_i} \partial_x\phi(t,x)\partial_x b_{i,d}(x)dx = \int_{a_i}^{b_i} \rho_q(t,x)b_{i,d}(x)dx \tag{4.96}$$

can be solved individually. The remaining problem is then the definition of the boundary conditions of the local problem (4.96). The integral introduces two integration

constants which cannot be defined using only the local information. Instead I propose solving the local problem defined by equation (4.96) with Dirichlet boundary conditions. The resulting spline is noted $S_{D,i}(x)$. Once this is done the resulting boundary conditions are shared with the other threads. A correction corresponding to the addition of a linear function $m_i x + p_i$ can then be calculated to find the final spline representation:

$$S_{C,i}(x) = S_{D,i}(x) + m_i x + p_i \tag{4.97}$$

This is done by solving the following system of equations:

$$S_{C,i}(b_i) = m_i b_i + p_i = m_{i+1} a_{i+1} + p_{i+1} = S_{C,i+1}(a_{i+1}) \qquad \forall 0 \le i < n_s \tag{4.98}$$

$$\partial_x S_{D,i}(b_i) + m_i = \partial_x S_{D,i+1}(a_{i+1}) + m_{i+1} \qquad \forall 0 \le i < n_s \tag{4.99}$$

where periodic boundary conditions imply that all objects indexed with n_s are equivalent to the same objects indexed with 0. As the number of domains n_s will usually be small, this system of equations should form a small matrix equation which will be easy to solve. The matrix system is of size $2n_s \times 2n_s$. For example in the case where there are three subdomains the matrix equation is:

$$\begin{pmatrix} b_0 & 1 & -b_0 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & \\ 0 & 0 & b_1 & 1 & -b_1 & -1 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ -a_0 & -1 & 0 & 0 & b_2 & 1 \\ -1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} m_0 \\ p_0 \\ m_1 \\ p_1 \\ m_2 \\ p_2 \end{pmatrix} = \begin{pmatrix} 0 \\ \partial_x S_{D,1}(a_1) - \partial_x S_{D,0}(b_0) \\ 0 \\ \partial_x S_{D,2}(a_2) - \partial_x S_{D,1}(b_2) \\ 0 \\ \partial_x S_{D,0}(a_0) - \partial_x S_{D,1}(b_2) \end{pmatrix} \tag{4.100}$$

In this periodic case, the matrix is singular. The final row must therefore be replaced with an alternative equation. Usually this equation is used to enforce the fact that the integral of the function over the periodic domain should be equal to zero.

$$\sum_{i=0}^{n_s-1} \int_{a_i}^{b_i} S_{D,i}(x) + m_i x + p_i \, dx = \sum_{i=0}^{n_s-1} \int_{a_i}^{b_i} S_{D,i}(x) \, dx + \frac{m_i \left(b_i^2 - a_i^2\right)}{2} + p_i (b_i - a_i) = 0 \tag{4.101}$$

This means that a total of 3 elements from each of the n_s sub-domains (the integral, the left-hand derivative, and the right-hand derivative) must be shared with each of the other sub-domains.

The final result can either be given point-wise or in spline form. Point-wise at points x_i, the result is obtained by evaluating the spline $S_{D,i}(x)$ at the chosen points and adding a correction:

$$\tilde{\phi}(x_i) = S_{D,i}(x_i) + m_i x_i + p \tag{4.102}$$

$$\partial_x \tilde{\phi}(x_i) = \partial_x S_{D,i}(x_i) + m \tag{4.103}$$

where $\tilde{\phi}(x_i)$ is the calculated approximation of the electric potential at the requested point x_i.

In spline form, the result is obtained by correcting the coefficients of the local splines. The coefficients c_{Ci} of the corrected spline $S_{C,i}(x)$ are equal to the sum of the coefficients c_{Di} of the spline respecting Dirichlet conditions $S_{D,i}(x)$ and coefficients c_{Li} of a spline, represented on the same basis, which interpolates the linear function $m_i x + p_i$:

$$c_{Ci} = c_{Di} + c_{Li}$$

Fortunately, as explained in Theorem 4, the latter coefficients can be calculated trivially using equation (4.105), without requiring the resolution of a second matrix equation.

Theorem 4. *Let $S(x)$ be a spline expressed on n_b basis splines of degree $d \geq 1$ respecting partition of unity, denoted $\{b_{0,d}(x), \dots, b_{n_b-1,d}(x)\}$:*

$$S(x) = \sum_{i=0}^{n_b-1} c_i b_{i,d}(x) \tag{4.104}$$

where c_i are the coefficients of the spline, and the basis is expressed on a series of knots $k_0 \leq \cdots \leq k_{n_b+d}$.

The coefficients of the spline that interpolates the linear function $f(x) = mx + p$ are equal to the value of the function at the Greville abcissae $g_{i,d}$:

$$c_i = m g_{i,d} + p \qquad \forall 0 \leq i < n_b \tag{4.105}$$

where the Greville abcissae are defined as the knot averages:

$$g_{i,d} = \sum_{j=1}^{d} \frac{k_i}{d} \tag{4.106}$$

Proof.

$$S(x) = \sum_{i=0}^{n_b-1} (m g_{i,d} + p) b_{i,d}(x) \tag{4.107}$$

$$= p + m \sum_{i=0}^{n_b-1} g_{i,d} b_{i,d}(x) \tag{4.108}$$

$$= p + m \sum_{i=0}^{n_b-1} g_{i,d} \left[\frac{x - k_i}{k_{i+d} - k_i} b_{i,d-1}(x) + \left(1 - \frac{x - k_{i+1}}{k_{i+d+1} - k_{i+1}}\right) b_{i+1,d-1}(x) \right] \tag{4.109}$$

$$= p + m g_{0,d} \frac{x - k_0}{k_d - k_0} b_{0,d-1}(x) + m g_{n_b-1,d} \left(1 - \frac{x - k_{n_b}}{k_{n_b+d} - k_{n_b}}\right) b_{n_b,d-1}(x)$$
$$+ m \sum_{i=1}^{n_b-1} \left[g_{i,d} \frac{x - k_i}{k_{i+d} - k_i} + g_{i-1,d} \left(1 - \frac{x - k_i}{k_{i+d} - k_i}\right) \right] b_{i,d-1}(x) \tag{4.110}$$

Using the fact that $b_{0,d-1}(x)$ is defined on $[k_0, k_d]$ and that $k_0 \leq k_d = a$, and that $b_{n_b+1,d-1}(x)$ is defined on $[k_{n_b}, k_{n_b+d}]$ and that $b = k_{n_b} \leq k_{n_b+d}$, we can neglect these

terms which are outside the spline that we consider.

$$S(x) = p + m \sum_{i=1}^{n_b-1} \left[\frac{x}{k_{i+d} - k_i} (g_{i,d} - g_{i-1,d}) + g_{i-1,d} - \frac{k_i}{k_{i+d} - k_i} (g_{i,d} - g_{i-1,d}) \right] b_{i,d-1}(x) \tag{4.111}$$

The definition of the greville abcissae implies the following relationship:

$$d(g_{i,d} - g_{i-1,d}) = d \sum_{j=1}^{d} \left(\frac{k_{i+j}}{d} - \frac{k_{i+j-1}}{d} \right) = k_{i+d} - k_i \tag{4.112}$$

Which we now use to simplify the equation and prove the theorem:

$$S(x) = p + m \sum_{i=1}^{n_b-1} \left[\frac{x}{d(g_{i,d} - g_{i-1,d})} (g_{i,d} - g_{i-1,d}) + g_{i-1,d} - \frac{k_i}{d(g_{i,d} - g_{i-1,d})} (g_{i,d} - g_{i-1,d}) \right] b_{i,d-1}(x) \tag{4.113}$$

$$= p + m \frac{x}{d} \sum_{i=1}^{n_b-1} b_{i,d-1}(x) + m \sum_{i=1}^{n_b-1} \frac{\sum_{j=1}^{d} k_{i-1+j}}{d} - \frac{k_i}{d} b_{i,d-1}(x) \tag{4.114}$$

$$= p + m \frac{x}{d} + m \sum_{i=1}^{n_b-1} \frac{\sum_{j=1}^{d-1} k_{i+j}}{d} b_{i,d-1}(x) \tag{4.115}$$

$$= p + m \frac{x}{d} + m \frac{d-1}{d} \sum_{i=1}^{n_b-1} g_{i,d-1} b_{i,d-1}(x) \tag{4.116}$$

$$= p + m \frac{x}{d} + m \frac{(d-1)x}{d} = mx + p \tag{4.117}$$

□

4.5. Conclusion

In this chapter I presented a new approach to local spline interpolation. I showed that semi-Lagrangian advection on these local splines is stable. Furthermore I described the steps necessary to create a parallel Vlasov-Poisson simulation using MPI parallelism. The next step is to implement the described simulation to verify the scaling properties of the described scheme and ensure that the loss of smoothness of the solution does not negatively affect the simulation results.

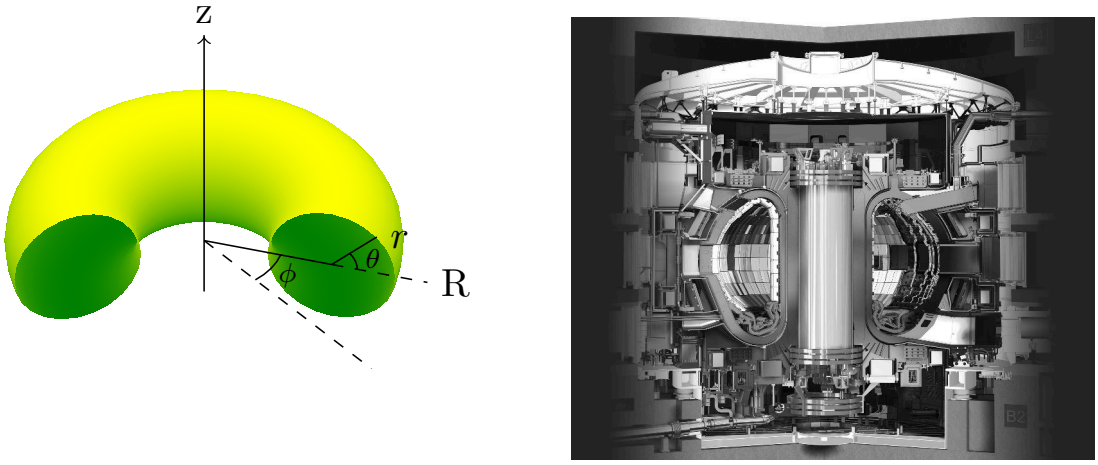
In Chapter 3, I showed that non-uniform splines are significantly slower than uniform splines, especially if GPUs are not available. This work provides a solution to that problem as different refinements can be used on different local splines. This allows the simulation to have the best of both worlds by, not only using fast uniform splines, but also having different refinements to ensure that steep gradients are fully resolved.

In the context of the GYSELA code (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016), this method is also useful as it provides an effective parallelisation method. Currently, in order to use splines in a given direction, all data along that direction must be available in order to construct a global spline. As a result the GYSELA code copies and transposes data between processes (G. Latu, N. Crouseilles, V. Grandgirard, et al. 2007). Such MPI communication is costly. This method could allow some of these data transfers to be avoided, replacing them with the small communications required to define the ghost cells.

Compared to the local spline method presented by Nicolas Crouseilles, Guillaume Latu, and Sonnendrücker 2009, the method presented here uses similar ghost cells, but it uses an alternative method to define them. It is simpler than the method presented by Nicolas Crouseilles, Guillaume Latu, and Sonnendrücker 2009, and can be used for splines of any degree not divisible by two, instead of only degree three. Furthermore the method works for non-uniform splines in addition to uniform splines. The stability of this method was proven, while that of the method presented by Nicolas Crouseilles, Guillaume Latu, and Sonnendrücker 2009 was not. The local spline method presented by Nicolas Crouseilles, Guillaume Latu, and Sonnendrücker 2009 was shown to work in two dimensions. In this chapter I have restricted myself to one dimensional problems, however the theory can easily be extended to two dimensions. The implementation and tests of such a 2D version of these local splines is left for future work.

5. Realistic geometry in GYSELA

The GYSELA code (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016) was originally designed with toroidal geometry as shown in Figure 5.1a. A toroidal geometry is defined by three coordinates (r,θ,φ), where r is the radial coordinate, and θ and φ are respectively the poloidal and toroidal angles. This geometry has a circular cross-section. A circle is a good approximation of the magnetic flux surfaces in the core of the plasma, near the centre of the tokamak. However as the development of GYSELA has progressed, improvements to the code have pushed the simulated domain closer and closer to the boundary. In this region the circle is a very poor approximation of the geometry of most tokamaks whose cross-section is more accurately described as D-shaped, as shown in Figure 5.1b. The Tore-Supra machine at IRFM/CEA used to be an exception to this rule as it had a circular geometry. However as part of its upgrade in 2013, a tungsten wall and a divertor were installed, changing this geometry. The tokamak has since been renamed WEST.



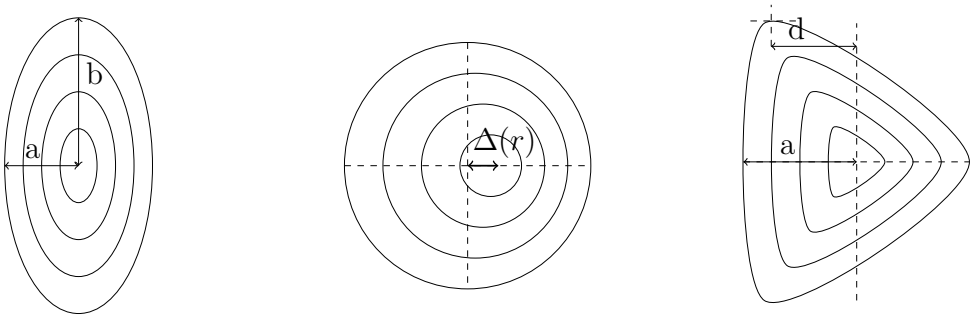
(a) The coordinates and geometry originally used in the GYSELA code.

(b) The shape of ITER. Source: https://www.iter.org/mach.

Figure 5.1.: A comparison of the geometry originally used in the GYSELA code and the actual shape of ITER

The D-shaped geometry is characterised by an elongation, a Shafranov shift, and a triangularity. Figure 5.2 shows how each of these deformations affects a geometry. The elongation is defined as $\kappa = \frac{b}{a}$. The triangularity δ is defined as $\delta = \frac{d}{a}$. It is important to take each of these characteristics into consideration as they can have an effect on the

turbulence in the plasma (Riva, Lanti, Sébastien Jolliet, et al. 2017; Angelino, X. Garbet, L. Villard, et al. 2009; Marinoni, Brunner, Camenen, et al. 2009). In order to do so the definition of the geometry in GYSELA must be updated.



(a) Elongation. (b) Shafranov shift. (c) Triangularity.

Figure 5.2.: The different types of deformation required for the D-shaped geometry.

There are several options for how to define the geometry. The ORB5 code (S. Jolliet, Bottino, Angelino, et al. 2007) is coupled to the CHEASE code (Lütjens, Bondeson, and Sauter 1996) which calculates the axisymmetric ideal MHD equilibria. The numerical equilibrium calculated with the simulation parameters is then used to define the geometry. This leads to an excellent approximation of the equilibrium, however it is a costly solution. The GENE code has two possible operating modes. It can either use the CHEASE code like ORB5 (Lapillonne, Dannert, Brunner, et al. 2008), or it can use the analytical Miller geometry (Mikkelsen, Howard, White, et al. 2018; Miller, Chu, Greene, et al. 1998). The solution chosen for the GYSELA code is to use an analytical geometry which can be calculated at the initialisation of the simulation. The geometry chosen is not the Miller geometry, but a geometry known as the "Culham geometry" (Connor, Cowley, Hastie, et al. 1988). This geometry will be presented in Section 5.1. The definitions of the various terms used are derived in Appendix B, using the techniques originally investigated by an intern, Guillaume Ferrière.

In Section 5.2 I present the changes that are made to the code in order to use this geometry. The new geometry is not orthogonal. This causes two major changes to the code. Firstly, all equations in the code must be expressed in generalised coordinates to ensure that new terms, are taken into account. Secondly the Poisson solver, which was previously based on the orthogonality assumption, must be replaced. Changes to the Poisson solver were made with the help of Kevin Obrejan. Finally, in Section 5.3 some preliminary results from GAM simulations, showing the successful implementation of the new geometry, are presented. This work, which validates the new geometry, was carried out by multiple physicists and interns; namely: Ken Leleux, Kevin Obrejan, Baptiste Legouix, and Peter Donnel.

5.1. Culham Geometry

The geometry for tokamak simulations is chosen to follow magnetic flux surfaces as most of the equations are defined in relation to the magnetic field. For toroidally axisymmetric systems, the flux ψ of the magnetic field across a closed curve is a solution to the Grad-Shafranov equation:

$$\Delta\psi = R\frac{\partial}{\partial R}\left(\frac{1}{R}\frac{\partial\psi}{\partial R}\right) + \frac{\partial^2\psi}{\partial Z^2} = -\mu_0 R^2\frac{dp(\psi)}{d\psi} - I(\psi)\frac{dI(\psi)}{d\psi} \tag{5.1}$$

where R, and Z are cylindrical coordinates describing a tokamak as shown in Figure 5.1a, μ_0 is the magnetic constant, $p(\psi)$ is the plasma pressure and $I(\psi)$ is the plasma current.

Several possibilities exist for defining a geometry from the Grad-Shafranov equation. Cerfon and Freidberg 2010 have provided analytical solutions to the equation using truncated polynomials. However these solutions only provide a radial coordinate for the flux surface. Defining a poloidal coordinate from this geometry is non-trivial as it also describes the geometry outside the LCFS. Miller, Chu, Greene, et al. 1998 have provided formulae for determining the magnetic field whose field lines are described by a basic description of a D-shaped geometry. Connor, Cowley, Hastie, et al. 1988 have also provided an analytical solution to the equation. In the GYSELA code we have chosen to use the geometry described by Connor, Cowley, Hastie, et al. 1988 which we will refer to as the "Culham geometry".

The analytical expression for this geometry is:

$$\begin{aligned} R(r,\theta) &= r\cos(\theta) - E(r)\cos(\theta) + T(r)\cos(2\theta) - A(r)\cos(\theta) + \Delta(r) + R_0 \\ Z(r,\theta) &= r\sin(\theta) + E(r)\sin(\theta) - T(r)\sin(2\theta) - A(r)\sin(\theta) \end{aligned} \tag{5.2}$$

where $E(r)$, $T(r)$, and $\Delta(r)$ are functions controlling respectively the elongation, triangularity, and Shafranov shift, and $A(r)$ is a correction term. $\Delta(r)$ is directly equal to the Shafranov shift. Approximate relations between $E(r)$ and $T(r)$, and the elongation κ and the triangularity δ will be derived in Section 5.1.1. The coordinates which will be used in the code are (r,θ), these replace the previous circular coordinates which were also noted (r,θ) and are shown in Figure 5.1a.

We introduce the functions $f(r)$ and $g(r)$ which are used to note the components of the magnetic field $B(r)$ which is orthogonal to ∇r:

$$B(r) = \nabla\phi\times\nabla\psi(r) + I(r)\nabla\phi\psi'(r) = B_0R_0\left(f(r)\nabla\theta + g(r)\nabla\phi\right) \tag{5.3}$$

where ∇r, $\nabla\theta$ and $\nabla\phi$ are respectively vectors in the radial, poloidal and toroidal directions.

The safety factor $q(r)$ defines the ratio between the number of times the magnetic field wraps around the torus toroidally compared to poloidally. The definition of the classical safety factor $q(r)$ in the large aspect ratio, and the zero-th order approximation of the Grad-Shafranov equation in the small parameter $\varepsilon = \frac{a}{R}$, where a is the

minor radius of the tokamak, can be used to define a relation between $f(r)$ and $g(r)$. Appendix C.3 shows the details of the Grad-Shafranov approximation. The relation between $f(r)$ and $g(r)$ is expressed as:

$$f(r) = \zeta(r) g(r) \tag{5.4}$$

$$\zeta(r) = \frac{r}{q(r) R_0} \tag{5.5}$$

$$f(r) f'(r) + \frac{f(r)^2}{r} + g'(r) + \frac{\mu_0 p'(r)}{B_0^2} = 0 \tag{5.6}$$

where $q(r)$ is approximated by the following equation:

$$q(r) = q_0 + (q_a - q_0) r^2, \tag{5.7}$$

where $q_0 = q(0)$ and $q_a = q(a)$. We use the following definition of the plasma pressure:

$$p(r) = p_a + (p_0 - p_a) \left(1 - r^2\right)^\gamma, \tag{5.8}$$

where γ is a constant, and p_0 and p_a are the pressures at respectively $r = 0$ and $r = a$.

The definitions of $E(r)$, $T(r)$, $\Delta(r)$ and $A(r)$ are obtained using the first order approximation of the Grad-Shafranov equation in the small parameter $\varepsilon = \frac{a}{R}$, and a quasi-toroidal assumption (see Appendix B). The details of this calculation can be found in Appendix C.3.

$$A(r) = \frac{r^3}{8R_0^2} + \frac{r\delta(r)}{2R_0} - \frac{E(r)^2}{2r} - \frac{T(r)^2}{r} \tag{5.9}$$

$$E''(r) + \left(\frac{1}{r} + \frac{2f'(r)}{f(r)}\right) E'(r) - 3\frac{E(r)}{r^2} = 0, \tag{5.10}$$

$$T''(r) + \left(\frac{1}{r} + \frac{2f'(r)}{f(r)}\right) T'(r) - 8\frac{T(r)}{r^2} = 0, \tag{5.11}$$

$$\delta'(r) = \frac{1}{R_0 r^2 f(r)^2} \left(\int_0^r r' f(r')^2 dr' - \int_0^r \frac{2r'^2 \mu_0 p'(r')}{B_0^2} dr' \right) \tag{5.12}$$

The integration constants of the functions $E(r)$, $T(r)$ and $\delta(r)$ are defined using the constants C_E and C_T such that $E(a) = C_E$, $T(a) = C_T$, and $\delta(a) = 0$.

These equations are analytical, but cannot easily be solved analytically. Therefore Equations (6.35), (6.36), and (6.41) are solved using a fourth order Runge-Kutta method with the following initial conditions:

$$E(r_0) = r_0, \tag{5.13}$$

$$E'(r_0) = 1, \tag{5.14}$$

$$T(r_0) = r_0^2, \tag{5.15}$$

$$T'(r_0) = 2r_0, g(r_0) = 1. \tag{5.16}$$

The integrals required for the definition of the Shafranov shift are also calculated numerically.

The parameters used to define the geometry are summarised in Table 5.1 along with example values which are used to define the geometry shown in Figure 5.3. In the GYSELA code these parameters are inputs which can be chosen for each simulation.

E_a	0.25	T_a	0.1	q_0	0.8	q_a	0.7	γ	1.0
p_0	10^5	p_a	10^4	B_0	1.0	R_0	5.0		

Table 5.1.: Parameters used to define the Culham geometry from Equations (6.33) - (6.43) in Figure 5.3

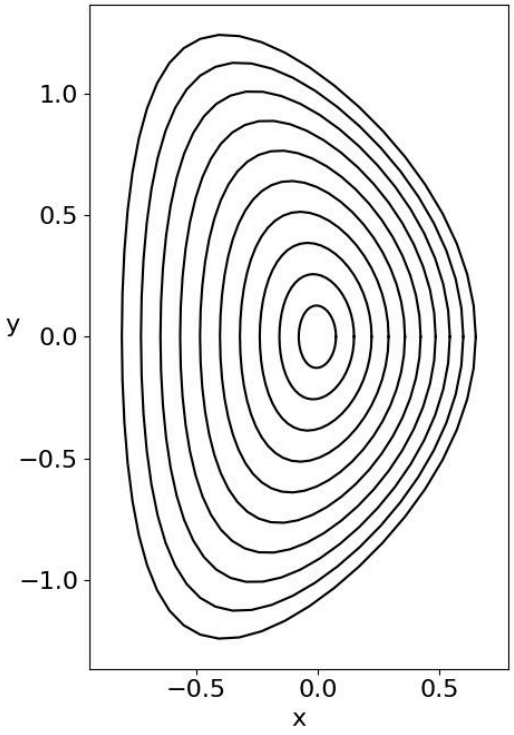


Figure 5.3.: Example of the magnetic field lines generated by the Culham geometry defined by the parameters in table 5.1

5.1.1. Geometric Properties

It is useful to have an approximation of the elongation and triangularity of this geometry.

These geometric properties are defined as shown in Figure 5.2. In order to obtain approximations we will neglect the correction term $A(r)$ which has a lower order than the other terms $\mathcal{O}(R_0\varepsilon^3) < \mathcal{O}(R_0\varepsilon^2)$. We also note that the Shafranov shift was chosen to be equal to 0 on the boundary.

It is therefore simple to estimate the relation between $E(a)$ and κ by examining a case with no triangularity:

$$\kappa = \frac{b}{a} \approx \frac{a\sin(\pi/2) + E(a)\sin(\pi/2)}{a\cos(0) - E(a)\cos(0)} = \frac{a + E(a)}{a - E(a)} \tag{5.17}$$

$$E(a) = a\frac{1-\kappa}{1+\kappa} \tag{5.18}$$

The calculation for triangularity is more complex. We will examine a case with no elongation to obtain the estimation. The triangularity is defined as (Sauter 2016):

$$\delta = \frac{1}{2}\left(\delta_{\text{top}} + \delta_{\text{bottom}}\right) = \frac{1}{2}\left(\frac{R_{\text{mid}} - R(Z = Z_{\max})}{r_{\text{avg}}} + \frac{R_{\text{mid}} - R(Z = Z_{\min})}{r_{\text{avg}}}\right) \tag{5.19}$$

where

$$R_{\text{mid}} = (R_{\max} + R_{\min})/2 = \frac{1}{2}\left(R_0 + a + T(a) + R_0 - a + T(a)\right) = R_0 + T(a) \tag{5.20}$$

and

$$r_{\text{avg}} = (R_{\max} - R_{\min})/2 = \frac{1}{2}\left(R_0 + a + T(a) - R_0 + a - T(a)\right) = a \tag{5.21}$$

The vertical component is an extremum when the following expression is satisfied:

$$\partial_\theta Z(a,\theta) = a\cos(\theta) - 2T(a)\cos(2\theta) = a\cos(\theta) - 4T(a)\cos^2(\theta) + 2T(a) = 0 \tag{5.22}$$

The positions where Z is extremal are therefore:

$$\cos(\theta) = \frac{a \pm \sqrt{a^2 + 32T(a)^2}}{8T(a)} = a\frac{1 \pm \sqrt{1 + 32\frac{T(a)^2}{a^2}}}{8T(a)} \tag{5.23}$$

The function $T(r)$ behaves similarly to r^2, we therefore rewrite this as:

$$\cos(\theta) \approx a\frac{1 \pm \sqrt{1 + 32\frac{T_a^2 a^4}{a^2}}}{8T_a a^2} = \frac{1 \pm \sqrt{1 + 32T_a^2 a^2}}{8T_a a} \tag{5.24}$$

The case where $\pm$ is $+$ can be ignored as it would give $\cos(\theta) > 1$ for any small $T_a a$. Remembering that:

$$\cos(2\theta) = \frac{\cos(\theta)}{2T(a)} \tag{5.25}$$

The triangularity can therefore be expressed as:

$$\begin{aligned}\delta =& \frac{1}{a}(R_0 + T(a) - R_0 - \cos(\theta) - T(a)\cos(2\theta)) \\ =& \frac{1}{a}\left(T(a) - \cos(\theta) - T(a)\frac{\cos(\theta)}{2T(a)}\right) \\ =& \frac{1}{a}\left(T(a) - \frac{3}{2}\cos(\theta)\right) \\ =& \frac{T(a)}{a} - \frac{3}{2}\frac{1-\sqrt{1+32\frac{T(a)^2}{a^2}}}{8T(a)}\end{aligned}$$

To simplify this expression further we use a Taylor expansion using the assumption that $32\frac{T(a)^2}{a^2} < 1$:

$$\delta = \frac{T(a)}{a} - \frac{3}{16T(a)}\left(1 - \left(1 + 16\frac{T(a)^2}{a^2} + \mathcal{O}(T(a)^4)\right)\right) \approx \frac{T(a)}{a} + \frac{3T(a)}{a^2} \tag{5.26}$$

These equations are usually considered in normalised coordinates, therefore $a = 1$, which leaves:

$$\delta \approx 4T(a) \tag{5.27}$$

$$T(a) \approx \frac{\delta}{4} \tag{5.28}$$

5.2. Modifications to GYSELA

Changing the underlying geometry of the equations should not change how they need to be solved. However when writing efficient code, simplifications are made based on the properties of the coordinates used in the simulation. In the case of cylindrical coordinates, several simplifications arise from the fact that the radial and poloidal coordinates are orthogonal.

In order to correctly model a system on non-orthogonal coordinates it is important to ensure that all equations are expressed correctly in generalised coordinates.

5.2.1. Generalised coordinates

An expression written in generalised coordinates is not written using vectors, but rather the components are used directly. I begin by a short overview of generalised coordinates and the formulae used for later calculations. These formulae can be found in a textbook on tensor calculus, for example (Sochi 2016a; Sochi 2016b). If a N-D vector $\vec{v}$ is expressed on the basis $\{\hat{e}_1, \ldots, \hat{e}_N\}$, it can be written as:

$$\vec{v} = \sum_{i=1}^{N} v^i \hat{e}_i \tag{5.29}$$

the components v^i are known as the *contravariant* components of the vector $\vec{v}$. The components v_i, defined as:

$$v_i = \vec{v} \cdot \hat{e}_i = \sum_{j=1}^{N} v^j \hat{e}_j \cdot \hat{e}_i, \tag{5.30}$$

are known as the *covariant* components.

On an orthonormal basis $\hat{e}_j \cdot \hat{e}_i = \delta_{ij}$, where δ_{ij} is the Kronecker symbol defined as:

$$\delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ otherwise} \end{cases} \tag{5.31}$$

However in a more general case this is not true. Instead the results of the scalar products of basis vectors are summarised in a matrix known as the *contravariant metric tensor*:

$$g = \begin{pmatrix} g_{11} & \dots & g_{1N} \\ \vdots & & \vdots \\ g_{N1} & \dots & g_{NN} \end{pmatrix} = \begin{pmatrix} \hat{e}_1 \cdot \hat{e}_1 & \dots & \hat{e}_1 \cdot \hat{e}_N \\ \vdots & & \vdots \\ \hat{e}_N \cdot \hat{e}_1 & \dots & \hat{e}_N \cdot \hat{e}_N \end{pmatrix} \tag{5.32}$$

The determinant $|g|$ of this matrix is also very useful.

This allows us to describe the relation between covariant and contravariant components:

$$v_i = \vec{v} \cdot \hat{e}_i = \sum_{j=1}^{N} (v^j \hat{e}_j) \cdot \hat{e}_i = \sum_{j=1}^{N} v^j g_{ij}. \tag{5.33}$$

We see that this is equivalent to a matrix multiplication. Thus we can define the covariant metric tensor elements g^{ij} such that:

$$v^j = \sum_{i=1}^{N} v_i g^{ij} \tag{5.34}$$

as the elements of the inverse of the contravariant metric tensor:

$$g^{-1} = \begin{pmatrix} g^{11} & \dots & g^{1N} \\ \vdots & & \vdots \\ g^{N1} & \dots & g^{NN} \end{pmatrix} = \begin{pmatrix} g_{11} & \dots & g_{1N} \\ \vdots & & \vdots \\ g_{N1} & \dots & g_{NN} \end{pmatrix}^{-1} \tag{5.35}$$

Contravariant components can be thought of as the elements of a column vector, while covariant components can be thought of as the elements of a row vector. This intuition shows us how covariant and contravariant components can be combined. It is possible to sum over elements, as long as one term in the element is covariant and the other is contravariant.

Expressions using covariant and contravariant components usually contain many summations. Therefore in order to declutter the notation, Einstein's convention is

frequently used. Einstein's convention says that all indices which appear on only one side of an equality are summed over. Thus with Einstein's convention a scalar product can be written as:

$$\vec{v} \cdot \vec{u} = v^i u_i \tag{5.36}$$

Finally to complete the notation we also mention the gradient. The contravariant component is denoted ∇^i, but it is more often often used in its covariant form:

$$\nabla \cdot \hat{e}_i = \partial_i = \frac{\partial}{\partial \xi^i} \tag{5.37}$$

where ξ^i is the covariant form of the i-th curvilinear coordinate associated with the direction $\hat{e}_i$.

With these definitions we now enumerate some useful equations which we will use in the rest of this chapter:

$$\nabla_j v_k = \partial_j v_k - v_i \Gamma_{k\ j}^{\ i} \tag{5.38}$$

$$\nabla_j v^k = \partial_j v^k + v^i \Gamma_{j\ i}^{\ k} \tag{5.39}$$

$$\nabla \cdot \vec{v} = \frac{1}{\sqrt{|g|}} \frac{\partial}{\partial \xi^i} \left(\sqrt{|g|} v^i \right) \tag{5.40}$$

$$\nabla^2 f = \frac{1}{\sqrt{|g|}} \frac{\partial}{\partial \xi^i} \left(\sqrt{|g|} g^{ik} \frac{\partial f}{\partial \xi^k} \right) \tag{5.41}$$

$$\Gamma_{i\ k}^{\ j} = \frac{1}{2} g^{jl} (\partial_k g_{il} + \partial_i g_{lk} - \partial_l g_{ki}) \tag{5.42}$$

$$\Gamma_{i\ k}^{\ i} = \frac{1}{\sqrt{|g|}} \partial_k \sqrt{|g|} \tag{5.43}$$

where $\vec{v}$ is a vector, and f is a scalar.

5.2.1.1. Generalised coordinates in GYSELA

Luckily GYSELA was originally designed with generalised coordinates in mind (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016). As a result the majority of the equations are already solved using the covariant and contravariant components of the necessary vectors.

In order to express these covariant and contravariant components on the new geometry the only additional information required is the definition of the metric tensor.

The new 2D geometry is defined on the unit vectors $\hat{e}_r = \nabla r$, and $\hat{e}_\theta = \nabla \theta$. The contravariant metric tensor is therefore defined as follows:

$$\begin{pmatrix} g_{rr} & g_{r\theta} \\ g_{\theta r} & g_{\theta\theta} \end{pmatrix} = \begin{pmatrix} |\nabla r|^2 & \nabla\theta \cdot \nabla r \\ \nabla\theta \cdot \nabla r & |\nabla\theta|^2 \end{pmatrix} \tag{5.44}$$

The definition of the Culham geometry given in Equation (6.33) is not invertible as

the terms $E(r)$, $T(r)$, $\Delta(r)$, and $A(r)$ cannot be expressed analytically, so a definition of $r(R,Z)$ and $\theta(R,Z)$ is not available. However the gradients can be deduced using the Jacobian of the system:

$$\begin{pmatrix} dR \\ dZ \end{pmatrix} = \begin{pmatrix} \partial_r R & \partial_\theta R \\ \partial_r Z & \partial_\theta Z \end{pmatrix} \begin{pmatrix} dr \\ d\theta \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} dr \\ d\theta \end{pmatrix} = \frac{1}{J} \begin{pmatrix} \partial_\theta Z & -\partial_\theta R \\ -\partial_r Z & \partial_r R \end{pmatrix} \begin{pmatrix} dR \\ dZ \end{pmatrix}$$

with $J = \partial_r R \partial_\theta Z - \partial_r Z \partial_\theta R$. The gradients are therefore defined as:

$$\begin{aligned} dr &= \frac{1}{J}(\partial_\theta Z\, dR - \partial_\theta R\, dZ) \Rightarrow \nabla r = \frac{1}{J}(\partial_\theta Z \nabla R - \partial_\theta R \nabla Z) \\ d\theta &= \frac{1}{J}(\partial_r R dZ - \partial_r Z dR) \Rightarrow \nabla \theta = \frac{1}{J}(\partial_r R \nabla Z - \partial_r Z \nabla R) \end{aligned} \tag{5.45}$$

The contravariant metric tensor is therefore defined as follows:

$$\frac{1}{J^2} \begin{pmatrix} \left((\partial_\theta Z)^2 + (\partial_\theta R)^2\right) & -(\partial_r R \partial_\theta R + \partial_r Z \partial_\theta Z) \\ -(\partial_r R \partial_\theta R + \partial_r Z \partial_\theta Z) & \left((\partial_r R)^2 + (\partial_r Z)^2\right) \end{pmatrix} \tag{5.46}$$

The determinant of this matrix is:

$$g = \frac{1}{J^4}\left((\partial_\theta Z)^2 + (\partial_\theta R)^2\right)\left((\partial_r R)^2 + (\partial_r Z)^2\right) - \frac{1}{J^4}(\partial_r R \partial_\theta R + \partial_r Z \partial_\theta Z)^2 \tag{5.47}$$

$$\begin{aligned} = &\frac{1}{J^4}\left[(\partial_r R)^2(\partial_\theta Z)^2 + (\partial_r R)^2(\partial_\theta R)^2 + (\partial_r Z)^2(\partial_\theta Z)^2 + (\partial_r Z)^2(\partial_\theta R)^2\right. \\ &\left. -(\partial_r R)^2(\partial_\theta R)^2 - 2\partial_r R \partial_\theta R \partial_r Z \partial_\theta Z - (\partial_r Z)^2(\partial_\theta Z)^2\right] \end{aligned} \tag{5.48}$$

$$= \frac{\left[(\partial_r R)^2(\partial_\theta Z)^2 - 2\partial_r R \partial_\theta R \partial_r Z \partial_\theta Z + (\partial_r Z)^2(\partial_\theta R)^2\right]}{J^2\left[(\partial_r R)^2(\partial_\theta Z)^2 - 2\partial_r R \partial_r Z \partial_\theta R \partial_\theta Z + (\partial_r Z)^2(\partial_\theta R)^2\right]} \tag{5.49}$$

$$= \frac{1}{J^2} \tag{5.50}$$

The covariant metric tensor is therefore defined as:

$$\begin{pmatrix} g^{rr} & g^{r\theta} \\ g^{\theta r} & g^{\theta\theta} \end{pmatrix} = \begin{pmatrix} \left((\partial_r R)^2 + (\partial_r Z)^2\right) & (\partial_r R \partial_\theta R + \partial_r Z \partial_\theta Z) \\ (\partial_r R \partial_\theta R + \partial_r Z \partial_\theta Z) & \left((\partial_\theta Z)^2 + (\partial_\theta R)^2\right) \end{pmatrix} \tag{5.51}$$

These components can easily be determined from Equation (6.33) and are described in Appendix C.

The only equation which was not already expressed in general coordinates was the quasi-neutrality equation. In this case the fact that

$$g_{r\theta} = g_{\theta r} = g^{r\theta} = g^{\theta r} = 0$$

was used to simplify the calculations. If this equality holds then the radial and poloidal directions can be treated independently. Therefore a FFT was used in the poloidal

direction, while FDM was used in the radial direction (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016). This equality does not hold in the case of the Culham geometry. The equation must therefore be expressed on general coordinates and must be solved using a 2D method.

5.2.2. Quasi-neutrality equation

The normalised quasi-neutrality equation in the case of kinetic electrons is:

$$-\frac{1}{n_{e0}}\sum_{s\in\{i,e\}} A_s \nabla_\perp \cdot \left(\frac{n_{s0}}{B_0}\nabla_\perp \phi\right) = \frac{\rho_{i1}}{n_{e0}} \tag{5.52}$$

where A_i is the normalised mass of the ions, n_{i0} and n_{e0} are respectively the normalised equilibrium density of the ions and electrons, B_0 is the amplitude of the magnetic field, ϕ is the electric potential, and ρ_{i1} is the normalised ion charge density perturbation.

The normalised quasi-neutrality equation in the case of adiabatic electrons is:

$$-\frac{1}{n_{e0}} A_i \nabla_\perp \cdot \left(\frac{n_{i0}}{B_0}\nabla_\perp \phi\right) + \frac{1}{Z_0^2 T_e}\left(\phi - \langle\phi\rangle_{FS}\right) = \frac{\rho_{i1}}{n_{e0}} \tag{5.53}$$

where Z_0 is the normalised charge, T_e is the normalised temperature of the electrons, and ρ_{i1} is the normalised ion charge density perturbation. $\nabla_\perp$ denotes the gradient perpendicular to the magnetic field defined as $\nabla - \nabla_\parallel = \nabla - \hat{b}\left(\hat{b}\cdot\nabla\right)$, where $\hat{b}$ is the unit vector in the direction of the magnetic field. The bracket $\langle\cdot\rangle_{FS}$ denotes the flux surface average. This value is defined as follows:

$$\langle\phi\rangle_{FS}(r) = \frac{\int \phi(r,\theta,\phi) J_x d\theta d\phi}{\int J_x d\theta d\phi} \tag{5.54}$$

where J_x is the normalised Jacobian of the coordinate system (r,θ,ϕ):

$$J_x = R(r,\theta) J(r,\theta) \tag{5.55}$$

where $J(r,\theta)$ is the Jacobian of the chosen 2D coordinates (r,θ) as defined in section 5.2.1. It is equal to $J = r$ in circular geometry, and is calculated for the Culham geometry in Appendix C.

We can use the following notation to describe both equation (5.53) and equation (5.53):

$$-\nabla_\perp \cdot \left(\alpha \nabla_\perp \phi\right) + \beta(\phi - \langle\phi\rangle_{FS}) = RHS \tag{5.56}$$

where for the kinetic electron case $\alpha = \sum_{s\in\{i,e\}} \frac{A_s n_{s0}}{B_0}$, $\beta = 0$, and $RHS = \rho_{i1}$, and for the adiabatic electron case $\alpha = \frac{A_i n_{i0}}{B_0}$, $\beta = n_{e0}/(Z_0^2 T_e)$, and $RHS = \rho_{i1}$.

Equation (5.53) is a 3D equation which is difficult to solve. In order to understand what simplifications may be made to allow solving this equation, we begin by expressing it in generalised coordinates.

In order to simplify the calculations, we begin by noting the following property:

$$\nabla_{\parallel} \cdot \hat{b} = 0 \tag{5.57}$$

Proof. Using the dot product rule:

$$\nabla (A \cdot B) = (A \cdot \nabla) B + (B \cdot \nabla) A + A \times (\nabla \times B) + B \times (\nabla \times A)$$

if $A = B = \hat{b}$ then:

$$\frac{1}{2}\nabla \left(\hat{b} \cdot \hat{b}\right) = 0 = \left(\hat{b} \cdot \nabla\right) \hat{b} + \hat{b} \times \left(\nabla \times \hat{b}\right) \implies \left(\hat{b} \cdot \nabla\right) \hat{b} = \left(\nabla \times \hat{b}\right) \times \hat{b}$$

where I have used the fact that $\hat{b} \cdot \hat{b} = 1$. We can now prove equation (5.57)

$$\nabla_{\parallel} \cdot \hat{b} = \hat{b}\left(\hat{b} \cdot \nabla\right) \cdot \hat{b} = \hat{b} \cdot \left(\hat{b} \cdot \nabla\right) \hat{b} = \hat{b} \cdot \underbrace{\left[\left(\nabla \times \hat{b}\right) \times \hat{b}\right]}_{\text{perpendicular to } \hat{b}} = 0$$

□

First let us consider the Laplacien:

$$\nabla_{\perp} \cdot \left(\alpha \nabla_{\perp} \phi\right) = \nabla_{\perp} \alpha \cdot \nabla_{\perp} \phi + \alpha \nabla_{\perp}^2 \phi \tag{5.58}$$

In order to write the equation in generalised coordinates we first need to rewrite the equation without $\nabla_{\perp}$ and $\nabla_{\parallel}$. I begin by considering the first term in equation (5.58):

$$\begin{aligned}
&\nabla_{\perp} \alpha \cdot \nabla_{\perp} \phi \\
=&\left[\nabla \alpha - \hat{b}\left(\hat{b} \cdot \nabla \alpha\right)\right] \cdot \left[\nabla \phi - \hat{b}\left(\hat{b} \cdot \nabla \phi\right)\right] \\
=&\nabla \alpha \cdot \nabla \phi - \nabla \alpha \cdot \hat{b}\left(\hat{b} \cdot \nabla \phi\right) - \left(\hat{b} \cdot \nabla \alpha\right) \hat{b} \cdot \nabla \phi + \hat{b} \cdot \hat{b}\left(\hat{b} \cdot \nabla \alpha\right)\left(\hat{b} \cdot \nabla \phi\right) \\
=&\nabla \alpha \cdot \nabla \phi - \left(\nabla \alpha \cdot \hat{b}\right)\left(\hat{b} \cdot \nabla \phi\right)
\end{aligned} \tag{5.59}$$

This term can be then expressed in generalised coordinates as:

$$\begin{aligned}
\nabla_{\perp} \alpha \cdot \nabla_{\perp} \phi =& (\nabla \alpha)^i (\nabla \phi)_i - \left((\nabla \alpha)_i b^i\right)\left(b^j (\nabla \phi)_j\right) \\
=& g^{ij} \frac{\partial \alpha}{\partial \xi^j} \frac{\partial \phi}{\partial \xi^i} - b^i b^j \frac{\partial \alpha}{\partial \xi^i} \frac{\partial \phi}{\partial \xi^j}
\end{aligned} \tag{5.60}$$

I now consider the second term in equation (5.58):

$$\begin{aligned}
&\alpha \nabla_{\perp}^2 \phi = \alpha \nabla_{\perp} \cdot \left[\nabla \phi - \hat{b}\left(\hat{b} \cdot \nabla \phi\right)\right] \\
=&\alpha \nabla_{\perp} \cdot \nabla \phi - \alpha \nabla_{\perp} \cdot \left[\hat{b}\left(\hat{b} \cdot \nabla \phi\right)\right] \\
=&\alpha \nabla_{\perp} \cdot \nabla \phi - \alpha \hat{b} \cdot \nabla_{\perp}\left(\hat{b} \cdot \nabla \phi\right) - \alpha\left(\hat{b} \cdot \nabla \phi\right) \nabla_{\perp} \cdot \hat{b}
\end{aligned}$$

$$
\begin{aligned}
&= \alpha\nabla^2\phi - \alpha\hat{b}\left(\hat{b}\cdot\nabla\right)\cdot\nabla\phi - \alpha\left(\hat{b}\cdot\nabla\right)\left(\hat{b}\cdot\nabla\phi\right) + \alpha\hat{b}\cdot\hat{b}\left(\hat{b}\cdot\nabla\right)\left(\hat{b}\cdot\nabla\phi\right) \\
&\quad - \alpha\left(\hat{b}\cdot\nabla\phi\right)\nabla\cdot\hat{b} \\
&= \alpha\nabla^2\phi - \alpha\hat{b}\cdot\left(\hat{b}\cdot\nabla\right)\nabla\phi - \alpha\left(\hat{b}\cdot\nabla\phi\right)\left(\nabla\cdot\hat{b}\right)
\end{aligned} \tag{5.61}
$$

This term can be then expressed in generalised coordinates as:

$$
\begin{aligned}
\alpha\nabla_\perp^2\phi &= \frac{\alpha}{\sqrt{|g|}}\frac{\partial}{\partial\xi^i}\left(\sqrt{|g|}g^{ik}\frac{\partial\phi}{\partial\xi^k}\right) - \alpha b^k b^j \nabla_j(\nabla\phi)_k - \alpha b^i(\nabla\phi)_i\frac{1}{\sqrt{|g|}}\frac{\partial}{\partial\xi^j}\left(\sqrt{|g|}b^j\right) \\
&= \frac{\alpha}{\sqrt{|g|}}\frac{\partial}{\partial\xi^i}\left(\sqrt{|g|}g^{ik}\frac{\partial\phi}{\partial\xi^k}\right) - \alpha b^k b^j\left(\frac{\partial^2\phi}{\partial\xi^j\partial\xi^k} - \frac{\partial\phi}{\partial\xi^i}\Gamma_k{}^i{}_j\right) \\
&\quad - \frac{\alpha}{\sqrt{|g|}}b^i\frac{\partial\phi}{\partial\xi^i}\frac{\partial}{\partial\xi^j}\left(\sqrt{|g|}b^j\right)
\end{aligned} \tag{5.62}
$$

Equation (5.56) can therefore be written in generalised coordinates as:

$$
\begin{aligned}
&b^i b^j\frac{\partial\alpha}{\partial\xi^i}\frac{\partial\phi}{\partial\xi^j} - g^{ij}\frac{\partial\alpha}{\partial\xi^j}\frac{\partial\phi}{\partial\xi^i} - \frac{\alpha}{\sqrt{|g|}}\frac{\partial}{\partial\xi^i}\left(\sqrt{|g|}g^{ik}\frac{\partial\phi}{\partial\xi^k}\right) \\
&+ \alpha b^k b^j\left(\frac{\partial^2\phi}{\partial\xi^j\partial\xi^k} - \frac{\partial\phi}{\partial\xi^i}\Gamma_k{}^i{}_j\right) + \frac{\alpha}{\sqrt{|g|}}b^i\frac{\partial\phi}{\partial\xi^i}\frac{\partial}{\partial\xi^j}\left(\sqrt{|g|}b^j\right) + \beta\phi - \beta\langle\phi\rangle_{FS} = RHS
\end{aligned}
$$

In GYSELA this equation is simplified by taking the first order approximation of $\hat{b} = \hat{e}_\varphi + O(\varepsilon^2)$. This implies that $b^r = b^\theta = 0$ which allows us to deduce b^φ using the fact that $\hat{b}$ is a unit vector:

$$
b^\varphi b_\varphi = b^\varphi b^\varphi g_{\varphi\varphi} = 1 \implies b^\varphi = \frac{1}{\sqrt{g_{\varphi\varphi}}}
$$

The equation can finally be written as:

$$
\begin{aligned}
&\frac{1}{g_{\varphi\varphi}}\frac{\partial\alpha}{\partial\varphi}\frac{\partial\phi}{\partial\varphi} - g^{ij}\frac{\partial\alpha}{\partial\xi^j}\frac{\partial\phi}{\partial\xi^i} - \frac{\alpha}{\sqrt{|g|}}\frac{\partial}{\partial\xi^i}\left(\sqrt{|g|}g^{ik}\frac{\partial\phi}{\partial\xi^k}\right) \\
&+ \frac{\alpha}{g_{\varphi\varphi}}\left(\frac{\partial^2\phi}{\partial\varphi^2} - \frac{\partial\phi}{\partial\xi^i}\Gamma_\varphi{}^i{}_\varphi\right) + \frac{\alpha}{\sqrt{|g|}}\frac{1}{\sqrt{g_{\varphi\varphi}}}\frac{\partial\phi}{\partial\varphi}\frac{\partial}{\partial\varphi}\left(\sqrt{\frac{|g|}{g_{\varphi\varphi}}}\right) + \beta\phi - \beta\langle\phi\rangle_{FS} = RHS
\end{aligned} \tag{5.63}
$$

5.2.2.1. Circular geometry

Up to now GYSELA used a circular geometry for the poloidal cross-section. This geometry is defined on the curvilinear coordinates (r,θ,φ), and has the following metric tensor:

$$
g(r) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & r^2 & 0 \\ 0 & 0 & R^2 \end{pmatrix} \qquad g^{-1}(r) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{r^2} & 0 \\ 0 & 0 & \frac{1}{R^2} \end{pmatrix} \tag{5.64}
$$

The determinant $|g| = r^2R^2$. With this geometry the Christoffel symbol $\Gamma_{\varphi\ \varphi}^{\ i}$ is defined as:

$$\Gamma_{\varphi\ \varphi}^{\ i} = \frac{1}{2}g^{il}(\partial_\varphi g_{\varphi l} + \partial_\varphi g_{l\varphi} - \partial_l g_{\varphi\varphi}) = -\frac{1}{2}g^{il}\partial_l g_{\varphi\varphi} = -\frac{1}{2}g^{il}\partial_l R^2$$

We therefore have:

$$\Gamma_{\varphi\ \varphi}^{\ i} = -\frac{1}{2}\partial_r R(r,\theta)^2 \tag{5.65}$$

$$\Gamma_{\varphi\ \varphi}^{\ i} = -\frac{1}{2R^2}\partial_\theta R(r,\theta)^2 \tag{5.66}$$

$$\Gamma_{\varphi\ \varphi}^{\ i} = 0 \tag{5.67}$$

Equation (5.63) in circular geometry can therefore be written as:

$$\begin{aligned}
&\frac{1}{R^2}\frac{\partial\alpha}{\partial\varphi}\frac{\partial\phi}{\partial\varphi} - \frac{\partial\alpha}{\partial r}\frac{\partial\phi}{\partial r} - \frac{1}{r^2}\frac{\partial\alpha}{\partial\theta}\frac{\partial\phi}{\partial\theta} - \frac{1}{R^2}\frac{\partial\alpha}{\partial\varphi}\frac{\partial\phi}{\partial\varphi} - \frac{\alpha}{rR}\frac{\partial}{\partial r}\left(rR\frac{\partial\phi}{\partial r}\right) - \frac{\alpha}{rR}\frac{\partial}{\partial\theta}\left(rR\frac{1}{r^2}\frac{\partial\phi}{\partial\theta}\right) \\
&- \frac{\alpha}{rR}\frac{\partial}{\partial\varphi}\left(rR\frac{1}{R^2}\frac{\partial\phi}{\partial\varphi}\right) + \frac{\alpha}{R^2}\left(\frac{\partial^2\phi}{\partial\varphi^2} + \frac{1}{2}\frac{\partial\phi}{\partial r}\frac{\partial R}{\partial r} + \frac{1}{2R^2}\frac{\partial\phi}{\partial\theta}\frac{\partial R}{\partial\theta}\right) \\
&+ \frac{\alpha}{rR}\frac{1}{R^2}\frac{\partial\phi}{\partial\varphi}\frac{\partial}{\partial\varphi}\left(\sqrt{\frac{r^2R^2}{R^2}}\right) + \beta\phi - \beta\langle\phi\rangle_{FS} = RHS
\end{aligned}$$

which simplifies to:

$$\begin{aligned}
&-\frac{\partial\alpha}{\partial r}\frac{\partial\phi}{\partial r} - \frac{1}{r^2}\frac{\partial\alpha}{\partial\theta}\frac{\partial\phi}{\partial\theta} - \frac{\alpha}{r}\frac{\partial}{\partial r}\left(r\frac{\partial\phi}{\partial r}\right) - \frac{\alpha}{r^2}\frac{\partial^2\phi}{\partial\theta^2} + \frac{\alpha}{2R^2}\frac{\partial\phi}{\partial r}\frac{\partial R}{\partial r} + \frac{\alpha}{2R^4}\partial\phi\partial\theta\frac{\partial R}{\partial\theta} \\
&+ \beta\phi - \beta\langle\phi\rangle_{FS} = RHS
\end{aligned}$$

The terms $\frac{\alpha}{2R^2}\frac{\partial\phi}{\partial r}\frac{\partial R}{\partial r}$ and $\frac{\alpha}{2R^4}\partial\phi\partial\theta\frac{\partial R}{\partial\theta}$ are considered to be negligible so the expression becomes:

$$-\frac{\partial\alpha}{\partial r}\frac{\partial\phi}{\partial r} - \frac{1}{r^2}\frac{\partial\alpha}{\partial\theta}\frac{\partial\phi}{\partial\theta} - \frac{\alpha}{r}\frac{\partial}{\partial r}\left(r\frac{\partial\phi}{\partial r}\right) - \frac{\alpha}{r^2}\frac{\partial^2\phi}{\partial\theta^2} + \beta\phi - \beta\langle\phi\rangle_{FS} = RHS$$

We see that there are no terms dependent on φ. This is due to the simplification $\hat{b} \approx \hat{e}_\varphi$. We also note that there are no terms which contain derivatives in r and θ. This allows the equation to be solved using different methods in each direction. Namely FFT in θ and *FDM* in r were used.

5.2.2.2. Culham geometry

The metric tensor in the geometry described by Culham's equilibrium on the curvilinear coordinates (r,θ,φ) takes the following form:

$$g(r,\theta)=\begin{pmatrix} g_{rr} & g_{r\theta} & 0 \\ g_{r\theta} & g_{\theta\theta} & 0 \\ 0 & 0 & R^2 \end{pmatrix} \qquad g^{-1}(r,\theta)=\begin{pmatrix} g^{rr} & g^{r\theta} & 0 \\ g^{r\theta} & g^{\theta\theta} & 0 \\ 0 & 0 & \frac{1}{R^2} \end{pmatrix} \tag{5.68}$$

The determinant $|g|=R^2|\tilde{g}(r,\theta)|$, where $|\tilde{g}(r,\theta)|$ is the determinant of the submatrix $\tilde{g}(r,\theta)=\begin{pmatrix} g_{rr} & g_{r\theta} \\ g_{r\theta} & g_{\theta\theta} \end{pmatrix}$. With this geometry the Christoffel symbol $\Gamma_{\varphi\ \varphi}^{\ i}$ is defined as:

$$\Gamma_{\varphi\ \varphi}^{\ i}=\frac{1}{2}g^{il}(\partial_\varphi g_{\varphi l}+\partial_\varphi g_{l\varphi}-\partial_l g_{\varphi\varphi})=-\frac{1}{2}g^{il}\partial_l g_{\varphi\varphi}=-\frac{1}{2}g^{il}\partial_l R^2$$

Equation (5.63) in Culham geometry can therefore be written as:

$$\begin{aligned}
&\frac{1}{R^2}\frac{\partial\alpha}{\partial\varphi}\frac{\partial\phi}{\partial\varphi}-g^{rr}\frac{\partial\alpha}{\partial r}\frac{\partial\phi}{\partial r}-g^{r\theta}\frac{\partial\alpha}{\partial r}\frac{\partial\phi}{\partial\theta}-g^{\theta r}\frac{\partial\alpha}{\partial\theta}\frac{\partial\phi}{\partial r}-g^{\theta\theta}\frac{\partial\alpha}{\partial\theta}\frac{\partial\phi}{\partial\theta}-\frac{1}{R^2}\frac{\partial\alpha}{\partial\varphi}\frac{\partial\phi}{\partial\varphi}\\
&-\frac{\alpha}{R\sqrt{|\tilde{g}|}}\frac{\partial}{\partial r}\left(R\sqrt{|\tilde{g}|}g^{rr}\frac{\partial\phi}{\partial r}\right)-\frac{\alpha}{R\sqrt{|\tilde{g}|}}\frac{\partial}{\partial r}\left(R\sqrt{|\tilde{g}|}g^{r\theta}\frac{\partial\phi}{\partial\theta}\right)\\
&-\frac{\alpha}{R\sqrt{|\tilde{g}|}}\frac{\partial}{\partial\theta}\left(R\sqrt{|\tilde{g}|}g^{\theta r}\frac{\partial\phi}{\partial r}\right)-\frac{\alpha}{R\sqrt{|\tilde{g}|}}\frac{\partial}{\partial\theta}\left(R\sqrt{|\tilde{g}|}g^{\theta\theta}\frac{\partial\phi}{\partial\theta}\right)\\
&-\frac{\alpha}{R\sqrt{|\tilde{g}|}}\frac{\partial}{\partial\varphi}\left(R\sqrt{|\tilde{g}|}\frac{1}{R^2}\frac{\partial\phi}{\partial\varphi}\right)+\frac{\alpha}{R^2}\frac{\partial^2\phi}{\partial\varphi^2}-\frac{1}{2R^2}g^{rr}\frac{\partial}{\partial r}R^2-\frac{1}{2R^2}g^{r\theta}\frac{\partial}{\partial\theta}R^2-\frac{1}{2R^2}g^{r\theta}\frac{\partial}{\partial r}R^2\\
&-\frac{1}{2R^2}g^{\theta\theta}\frac{\partial}{\partial\theta}R^2+\frac{\alpha}{R^2\sqrt{|\tilde{g}|}}\frac{\partial\phi}{\partial\varphi}\frac{\partial}{\partial\varphi}\left(\sqrt{\frac{R^2|\tilde{g}|}{R^2}}\right)+\beta\phi-\beta\langle\phi\rangle_{FS}=RHS
\end{aligned} \tag{5.69}$$

Finally using the fact that none of the elements of the metric tensor depend on φ, this simplifies to:

$$\begin{aligned}
&-g^{rr}\frac{\partial\alpha}{\partial r}\frac{\partial\phi}{\partial r}-g^{r\theta}\frac{\partial\alpha}{\partial r}\frac{\partial\phi}{\partial\theta}-g^{\theta r}\frac{\partial\alpha}{\partial\theta}\frac{\partial\phi}{\partial r}-g^{\theta\theta}\frac{\partial\alpha}{\partial\theta}\frac{\partial\phi}{\partial\theta}\\
&-\frac{\alpha}{\sqrt{|\tilde{g}|}}\frac{\partial}{\partial r}\left(\sqrt{|\tilde{g}|}g^{rr}\frac{\partial\phi}{\partial r}\right)-\frac{\alpha}{\sqrt{|\tilde{g}|}}\frac{\partial}{\partial r}\left(\sqrt{|\tilde{g}|}g^{r\theta}\frac{\partial\phi}{\partial\theta}\right)-\frac{\alpha}{\sqrt{|\tilde{g}|}}\frac{\partial}{\partial\theta}\left(\sqrt{|\tilde{g}|}g^{\theta r}\frac{\partial\phi}{\partial r}\right)\\
&-\frac{\alpha}{\sqrt{|\tilde{g}|}}\frac{\partial}{\partial\theta}\left(\sqrt{|\tilde{g}|}g^{\theta\theta}\frac{\partial\phi}{\partial\theta}\right)-\frac{1}{2R^2}g^{rr}\frac{\partial}{\partial r}R^2-\frac{1}{2R^2}g^{r\theta}\frac{\partial}{\partial\theta}R^2-\frac{1}{2R^2}g^{r\theta}\frac{\partial}{\partial r}R^2\\
&-\frac{1}{2R^2}g^{\theta\theta}\frac{\partial}{\partial\theta}R^2+\beta\phi-\beta\langle\phi\rangle_{FS}=RHS
\end{aligned} \tag{5.70}$$

As for the circular geometry there are no terms dependent on φ and the terms

$\frac{1}{2R^2} g^{il}\partial_l R^2$ are neglected leaving:

$$
\begin{aligned}
&- g^{rr}\frac{\partial\alpha}{\partial r}\frac{\partial\phi}{\partial r} - g^{r\theta}\frac{\partial\alpha}{\partial r}\frac{\partial\phi}{\partial\theta} - g^{\theta r}\frac{\partial\alpha}{\partial\theta}\frac{\partial\phi}{\partial r} - g^{\theta\theta}\frac{\partial\alpha}{\partial\theta}\frac{\partial\phi}{\partial\theta} \\
&- \frac{\alpha}{\sqrt{|\tilde{g}|}}\frac{\partial}{\partial r}\left(\sqrt{|\tilde{g}|}g^{rr}\frac{\partial\phi}{\partial r}\right) - \frac{\alpha}{\sqrt{|\tilde{g}|}}\frac{\partial}{\partial r}\left(\sqrt{|\tilde{g}|}g^{r\theta}\frac{\partial\phi}{\partial\theta}\right) - \frac{\alpha}{\sqrt{|\tilde{g}|}}\frac{\partial}{\partial\theta}\left(\sqrt{|\tilde{g}|}g^{\theta r}\frac{\partial\phi}{\partial r}\right) \\
&- \frac{\alpha}{\sqrt{|\tilde{g}|}}\frac{\partial}{\partial\theta}\left(\sqrt{|\tilde{g}|}g^{\theta\theta}\frac{\partial\phi}{\partial\theta}\right) + \beta\phi - \beta\langle\phi\rangle_{FS} = RHS
\end{aligned}
\tag{5.71}
$$

5.2.2.3. Solving the Poisson equation

Now that we have all the tools necessary to write all the elements of equation (5.53) on the Culham geometry, I will explain how this equation is solved in the updated version of GYSELA.

The equation is solved using FEM on the $\mathcal{C}^1$ polar splines described in Section 2.4.1.4. The solver was developed by Zoni and Güçlü 2019 as part of the Selalib library SeLaLib Development Team 2018 which was designed to provide modules for GYSELA. It will be described in detail in Chapter 6, where it will be compared with two alternative solvers. All three solvers are designed to solve the following 2D equation:

$$\nabla\cdot\left(\alpha\nabla\phi\right) + \beta\phi = f \tag{5.72}$$

As we can see, this equation does not include the flux surface average. This is because including this while maintaining a performant solver is very difficult. In particular for matrix based methods the inclusion of the flux surface average destroys any sparsity patterns, significantly increasing the memory requirements and total FLOPs for the solver. For the case with kinetic electrons described by equation (5.52), the flux surface average does not appear, so solvers of this form are sufficient to solve the problem.

For the case with adiabatic electrons, the equation will be solved iteratively, using the following two equations describing a fixed point scheme:

$$-\nabla\cdot\left(\frac{n_{i0}}{B_0}\nabla\tilde{\phi}^{n+1}\right) = \frac{\rho_{i1}}{A_i} - \frac{n_{e0}}{Z_0^2 T_e A_i}\left(\tilde{\phi}^n - \langle\tilde{\phi}^n\rangle_{FS}\right) \tag{5.73}$$

$$-\nabla\cdot\left(\frac{n_{i0}}{B_0}\nabla\tilde{\phi}^{n+1}\right) + \frac{n_{e0}}{Z_0^2 T_e A_i}\tilde{\phi}^{n+1} = \frac{\rho_{i1}}{A_i} - \frac{n_{e0}}{Z_0^2 T_e A_i}\langle\tilde{\phi}^n\rangle_{FS} \tag{5.74}$$

where $\tilde{\phi}^n$ is the n-th approximation of ϕ calculated by the iterative solver. Equation (5.73) is efficient at converging low radial modes, while equation (5.74) is efficient at converging high radial modes. We therefore have two cases for the solver:

(A) Solving equation (5.73), with α, β, and f from equation (5.72) defined as:

$$\alpha = \frac{n_{i0}}{B_0} \qquad \beta = 0 \qquad f = \frac{\rho_{i1}}{A_i} - \frac{n_{e0}}{Z_0^2 T_e A_i}\left(\tilde{\phi}^n - \langle\tilde{\phi}^n\rangle_{FS}\right)$$

(B) Solving equation (5.74), with α, β, and f from equation (5.72) defined as:

$$\alpha = \frac{n_{i0}}{B_0} \qquad \beta = \frac{n_{e0}}{Z_0^2 T_e A_i} \qquad f = \frac{\rho_{i1}}{A_i} - \frac{n_{e0}}{Z_0^2 T_e A_i} \langle \tilde{\phi}^n \rangle_{FS}$$

The complete method for solving equation (5.53) is then:

1. Initialise by solving case (A)
2. Solve case (B), stop if method has converged
3. Solve case (B), stop if method has converged
4. Solve case (A)
5. Repeat from step 2 until convergence

The convergence is checked at the steps solving case (B) (2 and 3).

The convergence is determined using the L_2 norm:

$$\left\| \phi^{n+1} - \phi^n \right\|_2 \leq TOL \left\| \phi^{n+1} \right\|_2 \tag{5.75}$$

where TOL is the chosen tolerance.

The number of repetitions of each case (two for (B) and one for (A)) was chosen after testing to try to obtain the fastest convergence. Some results of these tests, conducted by Kevin Obrejan, can be seen in Figure 5.4. These results were obtained during the execution of the GYSELA code with a typical configuration. In order to minimise the number of iterations, the previous solution is used as an initial guess, allowing faster convergence of the conjugate gradient method. We see that of the three solutions presented, the case with 2 repetitions of step 2 is faster on average.

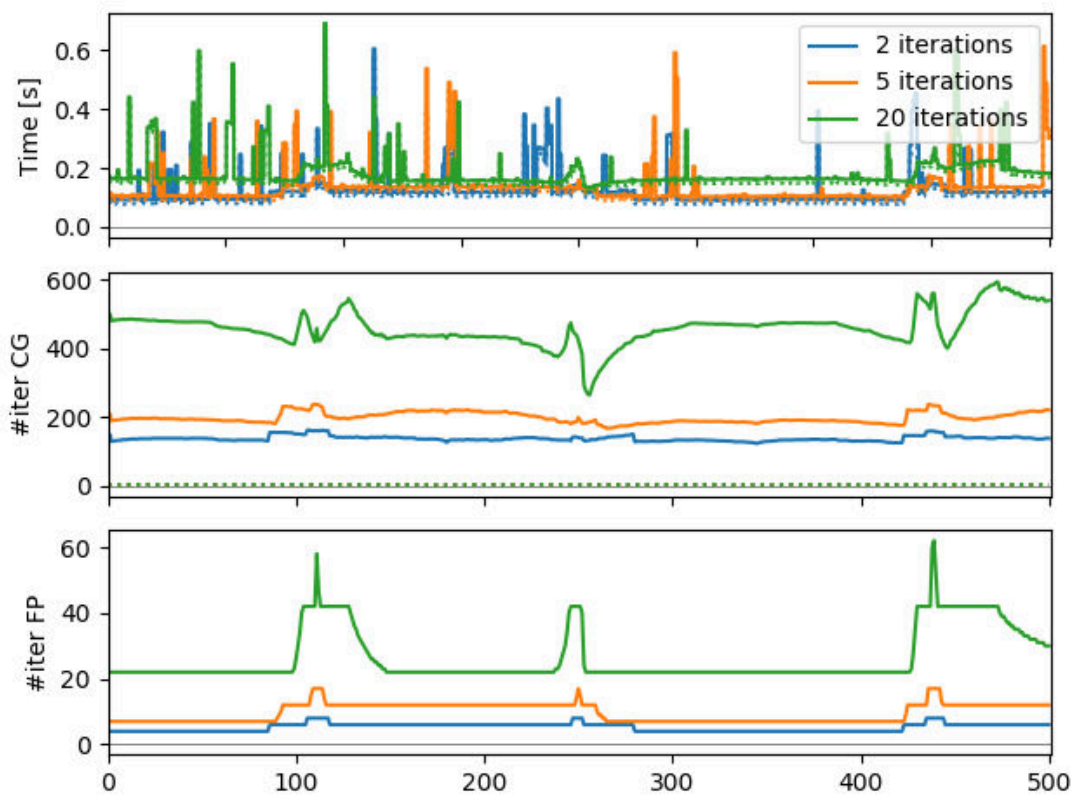


Figure 5.4.: Time, number of iterations of the conjugate gradient method, and number of iterations of the fixed point scheme for subsequent calls to the quasi-neutrality solver

5.3. GAM simulations with D-shaped geometry

In order to verify the implementation of the Culham geometry in GYSELA we will now study GAMs. In this section multiple simulations will be run with slightly different geometry. Unless otherwise specified the Culham geometry will be parameterised such that the Shafranov shift is 0 at the outer boundary, the outermost elongation κ is 1.0 (no elongation), and the outermost triangularity δ is 0.0 (no triangularity). The safety factor $q(r) = q_1$, which is a measure of the shape of the magnetic field, is set to 1.4. The other parameters necessary to define a simulation with the GYSELA code, as specified by V. Grandgirard, Abiteboul, J. Bigot, et al. 2016, are shown in Table 5.2.

When a plasma with homogeneous temperature and density is perturbed initially, the perturbed electric potential is Landau damped. The amplitude of the perturbation therefore decreases exponentially while oscillating, leaving only the homogeneous potential. This residual potential and the oscillations, known as GAMs, can be seen in the Fourier modes (Rosenbluth and Hinton 1998). The Fourier modes are denoted (m,n), where m is the index of the poloidal mode and n is the index of the toroidal mode. The oscillations describe the behaviour in the $(0,0)$ mode, but are observed

$\rho_\star$	$\frac{1}{160}$	A_s	1.0	Z_s	1.0	$\frac{R_0}{a}$	5.0	$\frac{r_{\text{int}}}{a}$	0.0	$\frac{r_{\text{ext}}}{a}$	1.0
Torus	1.0	nb_{vth0}	7.0	$\mu_{\max}$	12.0	q_1	1.4	q_2	0.0	q_3	1.
$\frac{r_{\text{peak}}}{a}$	0.5	κ_{ns_0}	10^{-7}	Δr_{ns_0}	0.2	κ_{Ts_0}	10^{-7}	Δr_{Ts_0}	0.1	$\frac{T_i}{T_e}$	1.0
κ	1.0	δ	0.0	N_r	256	N_θ	64	N_ϕ	16	$N_{v_{G\|}}$	128
N_μ	8										

Table 5.2.: Common parameters defined in V. Grandgirard, Abiteboul, J. Bigot, et al. 2016 used for GAM tests. The velocity phase space is defined by $-nb_{vth0}v_{Ts0} \leq v_{G\|} \leq nb_{vth0}v_{Ts0}$ and $0 \leq \mu \leq \mu_{\max}T_0/B_0$. Torus indicates the fraction of the torus simulated. The safety factor radial profile is defined as $q(r) = q_1 + q_2 exp(q_3 log(r/a))$. The radial density profile is defined by its gradient as $d\log n_{s0}(r)/dr = -\kappa_{ns_0}\cosh^{-2}\left((r - r_{\text{peak}}/a)/\Delta r_{ns0}\right)$. The same analytical expression is used for the temperature with κ_{Ts_0} and Δr_{Ts_0}.

in the $(0,0)$ and $(1,0)$ modes. The appearance in the $(1,0)$ mode is due to the charge separation which arises due to the vertical magnetic drift. Although the same effect occurs in the distribution function of both the electrons and the ions, as the potential depends on both, the $(1,0)$ mode is affected.

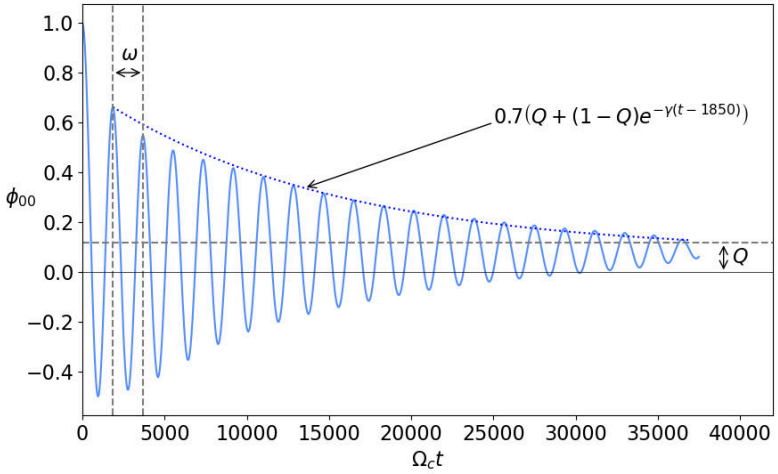
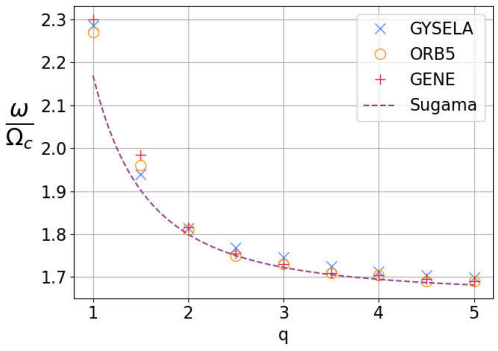


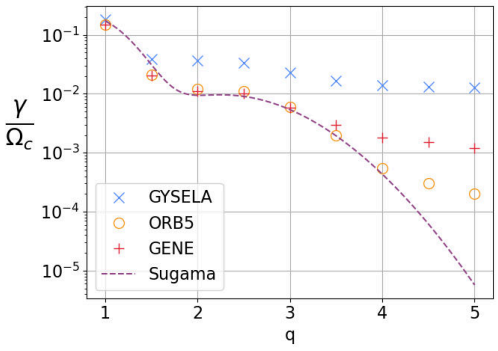
Figure 5.5.: Parameters characterising GAMs on simulations from GYSELA, run with $q = 1.5$, $E = 1.0$, and $T = 0.0$.

Figure 5.5 shows the evolution of the $(0,0)$ mode of the electric potential as a function of time. The time is normalised by the cyclotron frequency Ω_c. The oscillating behaviour seen is characterised by a frequency ω, a rate of exponential decay γ, and the residual Q. Sugama and Watanabe 2006 and Gao 2011 have provided theoretical predictions for the evolution of these characteristics. However both of these works rely on strong assumptions to simplify the problem which are not valid for a wide range of values (Biancalani, Bottino, Ehrlacher, et al. 2017). It is therefore more pertinent to

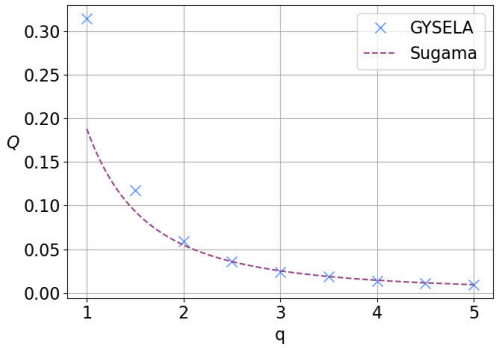
compare the results to other codes designed to solve similar problems, such as the Lagrangian PIC code ORB5 (S. Jolliet, Bottino, Angelino, et al. 2007), or the Eulerian code GENE (Jenko, Dorland, Kotschenreuther, et al. 2000). Values for these codes are provided in the work by Biancalani, Bottino, Ehrlacher, et al. 2017. Biancalani, Bottino, Ehrlacher, et al. 2017 also provided comparisons with the GYSELA code, however as a complex geometry had not yet been implemented they were only able to compare results with a dependence on geometrical parameters.



(a) The frequency of the GAM oscillations as a function of the safety factor q.



(b) The rate of damping γ of the GAM oscillations as a function of the safety factor q.

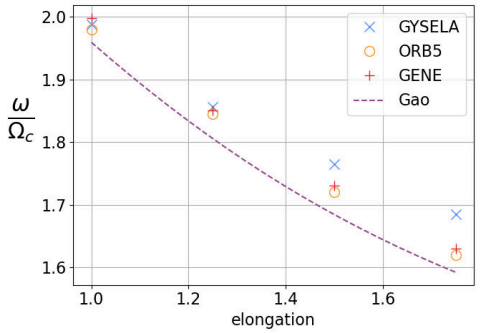


(c) The residual remaining after damping as a function of the safety factor q.

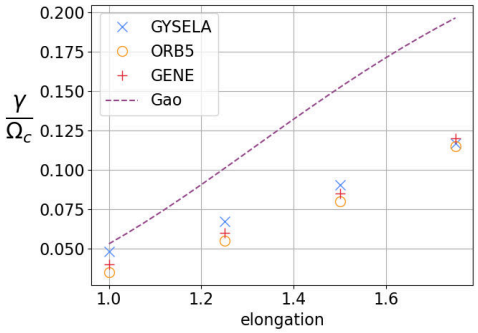
Figure 5.6.: Variation of GAM parameters as a function of the safety factor q for GYSELA, according to the theoretical results provided by Sugama, and where available ORB5 and GENE.

Sugama and Watanabe 2006 provide theoretical predictions for how the safety factor q influences the frequency, rate of decay, and residual. Figure 5.6 shows the variation of these parameters as a function of the safety factor for GYSELA, according to the theoretical results provided by Sugama, and where available ORB5 and GENE. This comparison had previously been carried out by Biancalani, Bottino, Ehrlacher, et al. 2017 using GYSELA's circular geometry. It is repeated here to validate the implementa-

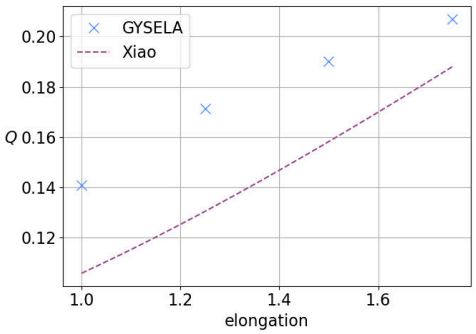
tion of the new geometry. We recall that even without elongation or triangularity, the new geometry is not equivalent to a circular geometry as it still contains a Shafranov shift and the correction term $A(r)$. We can see that the frequency and the residual are very close to the theoretical predictions. ORB5 and GENE also reproduce the analytical results for the frequency. In the case of the rate of decay, the results are less convincing. ORB5 and GENE produce results close to the theory for safety factors of less than 4, however there are differences for larger safety factors. In contrast GYSELA only manages to reproduce the expected profile for safety factors of 1.5 or less. Beyond this, the profile has the expected shape but not the expected magnitude. It should be noted that as Figure 5.6b is the only figure using a logarithmic scale, any differences in the results therefore appear more pronounced in this case.



(a) The frequency of the GAM oscillations as a function of the elongation.



(b) The rate of damping γ of the GAM oscillations as a function of the elongation.



(c) The residual remaining after damping as a function of the elongation.

Figure 5.7.: Variation of GAM parameters as a function of the elongation for GYSELA, according to the theoretical results provided by Gao and Xiao, and where available ORB5 and GENE.

Gao 2011 and Xiao and Catto 2006 provide theoretical predictions for how the elongation of the geometry influences the frequency, rate of decay, and residual. Figure 5.7 shows the variation of these parameters as a function of the elongation for GYSELA, according to the theoretical results provided by Gao, and where available

ORB5 and GENE. While the frequency results remain reasonably close to the theory, both the frequency and the damping rate results show the limitations of the theoretical approximations. In Figures 5.7a and 5.7b the numerical results are all in much closer agreement with one another than with the theory. As GYSELA produces results similar to those achieved with ORB5 and GENE this still allows the validation of the method. The results for the residual shown in Figure 5.7c, were unfortunately unavailable for ORB5 and GENE, however we can see that GYSELA's results are of the same order of magnitude as the analytical results and the overall gradient is similar. We do not expect to see a better correlation with the theory given the poor correlation shown in Figure 5.7b.

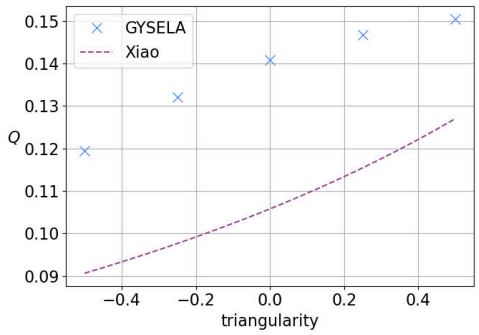


Figure 5.8.: The residual remaining after damping as a function of the triangularity for GYSELA, according to the theoretical results provided by Xiao.

Xiao and Catto 2006 also provide theoretical predictions for how the triangularity of the geometry influences the residual. Figure 5.8 shows the variation of the residual with the triangularity. Given that the theoretical result is based on the same simplifications as those used to obtain the estimations in Figure 5.7 we cannot expect a more precise result than the one obtained in Figure 5.7b. GYSELA's results are of the same order of magnitude as the analytical results and the overall gradient is similar, which suggests that the results are correct.

The preliminary results shown here for the Culham geometry with elongation, triangularity and Shafranov shift are encouraging. They therefore open the door to allow large simulations to be run with this geometry. For example the impact of the triangularity on turbulent transport can now be studied.

5.4. Conclusion

In this chapter a new realistic geometry, known as the "Culham geometry", was presented. This geometry is elongated with triangularity and a Shafranov shift. It is an analytical solution to the Grad-Shafranov equation which describes the magnetic flux surfaces in a plasma.

I derived the covariant and contravariant metric tensors for this geometry to allow the 5D Vlasov equation to be expressed in generalised coordinates. I also detailed the

definition of the Poisson equation in generalised coordinates for both circular geometry and "Culham geometry". This highlighted the possible choices of solver for this equation. As the new geometry requires a 2D solver, a new solver was implemented in GYSELA. This solver, and two alternatives will be discussed in the next chapter.

Finally analytical results concerning GAMs were compared to both the GYSELA code and results from ORB5 code (S. Jolliet, Bottino, Angelino, et al. 2007) and GENE (Jenko, Dorland, Kotschenreuther, et al. 2000), to show that the updated GYSELA code is capable of reproducing the expected physical behaviour. This new feature of the GYSELA code will now allow more complex studies to be conducted, for example studying the impact of the triangularity on turbulent transport.

6. Poisson Solver

6.1. Introduction

This chapter reproduces the work of Bourne, Leleux, Kormann, et al. 2022. In it we compare the Spline FEM solver used in Section 5.2.2 to two alternative solvers. These solvers may be useful, not only for the GYSELA code, but potentially for other gyrokinetic codes.

At each time step in gyrokinetic codes, one 5D Vlasov equation must be solved for each species, as well as a 3D Poisson-like equation describing the quasi-neutrality. The solution of the latter 3D system is very computationally expensive. While some codes, such as GENE-X (Michels, Stegmeir, Ulbl, et al. 2021) and EUTERPE (Hatzky, Tran, Könies, et al. 2002), solve this equation in its entirety, the majority of codes including GYSELA (Bouzat, Bressan, Virginie Grandgirard, et al. 2018), and ORB5 (S. Jolliet, Bottino, Angelino, et al. 2007) simplify the equation to a series of independent 2D equations. The 3D equation contains a derivative along the direction perpendicular to the magnetic field lines. These lines have a poloidal and a toroidal component, however in an axisymmetric configuration, it is possible to neglect the small poloidal component. This limits the configurations that can be simulated, thus GYSELA, and ORB5 can only simulate tokamaks while GENE-X and EUTERPE can also model stellarators.

In this chapter, we are interested in the solution of the 2D gyrokinetic Poisson-like equation $Lu = f$ with homogeneous Dirichlet boundary conditions, defined as:

$$\begin{aligned} Lu = -\nabla \cdot (\alpha \nabla u) + \beta u = f \text{ in } \Omega, \\ u = 0 \text{ on } \partial\Omega, \end{aligned} \tag{6.1}$$

where $\Omega \subset \mathbb{R}^2$ is a disk-like domain, $f : \Omega \to \mathbb{R}$ is the right hand side, $\alpha : \Omega \to \mathbb{R}$ is a non-constant coefficient involving the density profile, and $\beta : \Omega \to \mathbb{R}$ is a non-constant coefficient inversely proportional to the temperature profile. Three different solvers are compared, which use a variety of methods to solve Equation (6.1). The goal is to determine which solver is best adapted to this problem given the constraints of the framework where it will be implemented. In particular, we focus on an implementation in the GYSELA code (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016), however, we strive to present the advantages and disadvantages of each solver in a way that allows this comparison to be generalised to other codes. The three solvers and their implementations are described in detail in subsequent sections. They are a spline-based finite elements solver operating on polar coordinates, referred to as the Spline FEM

solver (Section 6.2), a geometric multigrid solver operating on polar coordinates, referred to as the GmgPolar solver (Section 6.3), and a finite differences solver operating on Cartesian coordinates with an embedded boundary approach, referred to as the Embedded Boundary solver (Section 6.4). The main similarities and differences of the solvers are summarised in Table 6.1.

	Spline FEM solver	GmgPolar solver	Embedded Boundary solver
Numerical Method	Finite Elements	Finite Differences	Finite Volumes
Linear equation solver	Conjugate Gradient	Multigrid	Multigrid
Singular Point	$\mathcal{C}^1$ polar splines	Handled in discretisation	N/A
Coordinates	Polar	Polar	Cartesian
Asymptotic accuracy	Degree dependent	Up to 4	Up to 2

Table 6.1.: Comparison of the main similarities and differences of the three solvers. Details about these results can be found in Sections 6.2-6.4 and the references therein.

The Spline FEM solver and the GmgPolar solver represent the domain using polar coordinates (r,θ), i.e. based on an invertible mapping from the Cartesian coordinates (x,y) to the polar coordinates $(r,\theta) \in (r_0,a] \times [0,2\pi)$, where r is the normalised radius, θ is the poloidal angle, r_0 is the minimum value of r, and the maximum value of r is the minor radius a of the torus describing the tokamak. The mapping is illustrated in Figure 6.1. The Cartesian coordinates are sometimes referred to as the "physical" coordinates, while the polar or curvilinear coordinates are known as the "logical" coordinates.

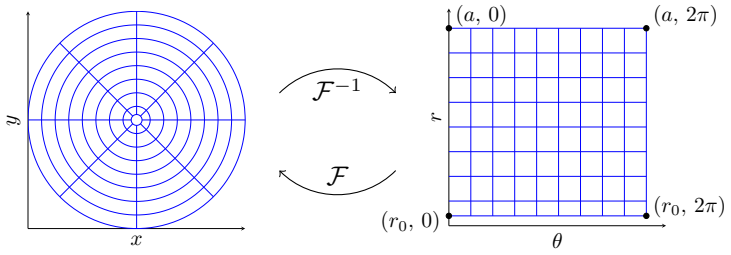


Figure 6.1.: The curvilinear coordinates defined by a mapping $\mathcal{F}$ between the Cartesian and polar coordinates $(r,\theta) \in [r_0,a] \times [0,2\pi)$.

Polar coordinates are most commonly used to describe a circle, however this is a poor representation of the cross-section of a tokamak. According to Connor, Cowley, Hastie, et al. 1988, the cross-section can be described by a disk to which multiple transformations are applied. These transformations elongate the disk, give it triangularity, or introduce a Shafranov shift. In this work, we will aim to describe the problem on the geometry described by Connor, Cowley, Hastie, et al. 1988, known as the "Culham geometry". The GYSELA code has recently been adapted to target this geometry in

order to take more realistic geometries into account. However, it is not possible to compute an analytical solution to Equation (6.1) on this geometry so it cannot be used to compute the error which arises when using each of the three solvers. In order to evaluate the comparative accuracy of the solvers, preliminary tests will be carried out on analytical geometries in Section 6.5.

The polar coordinates, used by the Spline FEM solver and the GmgPolar solver, provide a more natural representation of the geometries than the Cartesian coordinates used by the Embedded Boundary solver, but they give rise to two challenges. First, an artificial singularity is introduced at the origin of the mapping. This point is difficult to handle numerically, so many solvers choose a positive minimum radius $r_0 > 0$. However, there are important problems in magnetic fusion where a correct treatment of the pole is essential. Therefore, a solver targeting a plasma simulation code such as GYSELA should ideally handle this singularity. The Spline FEM solver employs $\mathcal{C}^1$ smooth polar splines as explained by Zoni and Güçlü 2019 to handle the singularity. The GmgPolar solver uses finite differences across the origin to avoid the issue, as detailed by Martin J Kühn, Kruse, and Rüde 2022.

The second challenge is due to the anisotropy which appears in the meshing of the (r,θ) plane. The finite elements scheme used in the Spline FEM solver uses the metric tensor to handle this anisotropy. The 9-point finite differences scheme used in the GmgPolar solver was constructed to naturally handle the use of anisotropic meshes. Both solvers can handle non-uniform meshes which allows the use of additional refinements to compensate for the anisotropy.

The chapter is organised as follows, in Section 6.2 the Spline FEM solver is introduced, the GmgPolar solver is introduced in Section 6.3, and the Embedded Boundary solver in Section 6.4. In Section 6.5, we compare these three approaches for the solution of the gyrokinetic Poisson-like equation on analytical test cases. In Section 6.6, we compare the behaviour of the three solvers on the non-analytical "Culham geometry". In Section 6.7, we discuss the difficulties encountered when tackling more complex geometries. The goal of this chapter is to give the advantages of each solver with a view to an integration in the GYSELA code.

6.2. Spline FEM

In Zoni and Güçlü 2019, the authors propose a B-spline finite element solver where, following the approach of isogeometric analysis, the geometry is described by a spline mapping. This solver uses specially constructed basis functions, proposed by Toshniwal, Speleers, Hiemstra, et al. 2017, around the singularity to ensure that a $\mathcal{C}^1$ smooth solution can be found. These basis functions are described in detail in Chapter 2.4.1.4. In this section, we summarise the scheme. For more details see Zoni and Güçlü 2019.

The grid upon which the solution evolves is constructed from the 2D spline representation, which is in turn constructed from a grid of break points. The grid of break points is defined as a cross product of $n_{cr}+1$ break points in the r-direction and $n_{c\theta}$ break points in the periodic θ-direction (see Figure 6.1). These break points can be

uniform or non-uniform. There are therefore $n_{cr}n_{c\theta}$ cells on the grid. 1D splines of degree d in the r-direction are defined on this grid by choosing knots such that

$$k_0 = k_1 = \cdots = k_d = b_0 < k_{d+1} = b_1 < \ldots k_{n_{cr}+d} = b_{n_{cr}+1} = k_{n_{cr}+d+1} = \cdots = k_{n_{cr}+2d}, \tag{6.2}$$

where k_i is the i-th knot, and b_i is the i-th break point; and 1D splines of degree d in the θ-direction are defined on the grid using periodic knots:

$$k_0 = b_{n_{c\theta}-d} < \cdots < k_{d-1} = b_{n_{c\theta}} < k_d = b_0 < \ldots$$
$$\cdots < k_{n_{c\theta}+d} = b_{n_{c\theta}} < k_{n_{c\theta}+d+1} = b_0 < \cdots < k_{n_{c\theta}+2d} = b$$

2D splines which do not handle the singular point are obtained using a basis defined as

$$B_l(r,\theta) = B_{in_{c\theta}+j}(r,\theta) = b_{i,d_r}(r)b_{j,d_\theta}(\theta), \tag{6.3}$$

where $B_l(r,\theta)$ is the l-th 2D basis function, $l = in_{c\theta} + j$, $b_{i,d_r}(r)$ is the i-th basis spline of degree d_r in the r-direction, and $b_{j,d_\theta}(\theta)$ is the j-th basis spline of degree d_θ in the θ-direction. In this work, the same degree is always used in the r-direction and the θ-direction, $d = d_r = d_\theta$. This results in $n_{br}n_{b\theta}$ basis functions where $n_{br} = n_{cr} + d_r$ is the number of basis functions in the non-periodic r-direction, and $n_{b\theta} = n_{c\theta}$ is the number of basis functions in the periodic θ-direction. There are $n_{br}n_{b\theta}$ interpolation points defined as the cross product of the Greville points (Farin 1993) of the splines in the r-direction and the θ-direction.

In the solver described by Zoni and Güçlü 2019, the smallest radial break point $r_0 = 0$ represents the singular point. In order to obtain a basis which is $\mathcal{C}^1$ at the singular point, the first $2n_{b\theta}$ basis functions

$$B_{in_\theta+j}(r,\theta) = b_{i,d_r}(r)b_{j,d_\theta}(r) \quad ,\forall i \in \{0,1\}, \forall 0 \leq j < n_{b\theta} \tag{6.4}$$

are replaced by three new basis functions $\hat{B}_l(r,\theta)$. The replaced basis functions are illustrated in Figure 6.2. The new basis functions $\{\hat{B}_0(r,\theta), \hat{B}_1(r,\theta), \hat{B}_2(r,\theta)\}$ are constructed such that they form a basis of a 2D bivariate polynomial of degree 1 at the singular point. The basis functions for the θ-direction splines which are used in this solver are therefore $\{\hat{B}_0(r,\theta), \hat{B}_1(r,\theta), \hat{B}_2(r,\theta)\}$ and

$$\hat{B}_{l+3}(r,\theta) = B_{l+2n_{b\theta}}(r,\theta) \quad ,\forall 0 \leq l < (n_{br}-2)n_{b\theta}. \tag{6.5}$$

The interpolation points remain the same. There are therefore $(n_{br}-1)n_{b\theta}+1$ interpolation points as the singular point only needs to be provided once.

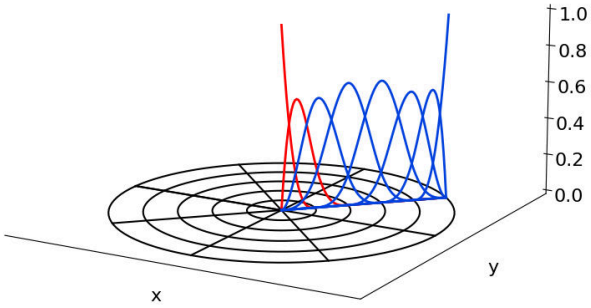


Figure 6.2.: The radial component of the 2D basis splines of degree 3. The basis-splines which are replaced by the C^1 polar basis functions are shown in red

With the chosen basis functions, Equation (6.1) can be solved using a finite elements solver by writing the equation in its weak form:

$$\int\int \left[\beta(r)u(r,\theta)\hat{B}_l(r,\theta) + \alpha(r)\nabla u(r,\theta)\cdot\nabla\hat{B}_l(r,\theta)\right]\mathcal{J}\,dr\,d\theta = \int\int f(r,\theta)\hat{B}_l(r,\theta)J\,dr\,d\theta, \quad (6.6)$$

where $\mathcal{J}$ is the determinant of the Jacobian matrix of the coordinate transformation. The solution $u(r,\theta)$, and the right hand side $f(r,\theta)$ can be expressed on the same basis functions, which leads to an expression of the form

$$u(r,\theta) = \sum_{l=0}^{n_{b\theta}(n_{br}-2)+3} u_l\hat{B}_l(r,\theta). \quad (6.7)$$

This allows the system to be expressed using the matrix equation

$$(M+S)\,\widehat{u} = M\widehat{f}, \quad (6.8)$$

where the vectors $\widehat{u}$ and $\widehat{f}$ contain the spline coefficients necessary to represent the functions u and f on the spline basis, the matrix M, known as the mass matrix, is defined as

$$M_{i,j} = \int\int \beta(r,\theta)\hat{B}_i(r,\theta)\hat{B}_j(r,\theta)\mathcal{J}\,dr\,d\theta, \quad (6.9)$$

and the matrix S, known as the stiffness matrix, is defined as

$$S_{i,j} = \int\int \alpha(r,\theta)\left[\sum_{\xi_1\in\{r,\theta\}}\sum_{\xi_2\in\{r,\theta\}} \frac{\partial\hat{B}_i}{\partial\xi_1}(r,\theta)g^{\xi_1\xi_2}\frac{\partial\hat{B}_j}{\partial\xi_2}(r,\theta)\right]\mathcal{J}\,dr\,d\theta. \quad (6.10)$$

where $g^{\xi_1\xi_2}$ is the scalar product between $\hat{e}_{\xi_1}$, the unit vector in the ξ_1 direction, and $\hat{e}_{\xi_2}$, the unit vector in the ξ_2 direction. In the case of an orthogonal coordinate system $g^{rr} = g^{\theta\theta} = 1$ and $g^{r\theta} = g^{\theta r} = 0$, however this is not true for an arbitrary coordinate

system (see Section 5.2.1 for details). The "Culham geometry" which will be considered in Section 6.6 is an example of a non-orthogonal coordinate system.

The ability to express our problem as a matrix equation is practical as there are many existing methods for solving such equations. In our work, we use a preconditioned conjugate gradient method. This is possible as the matrix describing the system is symmetric positive-definite. The preconditioner used is a Jacobi preconditioner. The matrix is quite large which could make it costly to store, however the properties of the basis splines allow us to reduce the storage significantly from $[(n_{br}-2)n_{b\theta}+3]^2 = \mathcal{O}\left(n_{b\theta}^2 n_{b\theta}^2\right)$ elements to $(n_{br}-2)n_{b\theta}(2\cdot d_r+1)(2d_\theta+1)+9+6n_\theta d_r = \mathcal{O}(n_{br}n_{b\theta}d_r d_\theta)$.

This reduction is done by remarking that each 1D b-spline only overlaps with d b-splines to its left, and d b-splines to its right. In order to use this fact the matrix defining the system $H = M + S$ is expressed in block-format:

$$H = \begin{pmatrix} H_1 & H_2 \\ H_3 & H_4 \end{pmatrix} \tag{6.11}$$

with $H_1 \in \mathbb{R}^{3\times 3}$ handling the terms containing only the central $\mathcal{C}^1$ basis splines, $H_2 \in \mathbb{R}^{3\times(n_{b\theta}(n_{br}-2))}$ and $H_3 \in \mathbb{R}^{(n_{b\theta}(n_{br}-2))\times 3}$ handling the terms containing both central $\mathcal{C}^1$ basis splines and the other splines, and $H_4 \in \mathbb{R}^{(n_{b\theta}(n_{br}-2))\times(n_{b\theta}(n_{br}-2))}$ handling the terms containing only basis splines defined as a product of 1D basis splines. The elements of H_4 are therefore expressed as:

$$H_4 = H_{4,i,j}(\hat{B}_i, \hat{B}_j) = H_{4,i,j}(b_{k,d_r}b_{l,d_\theta}, b_{m,d_r}b_{p,d_\theta}) = H_{4,k,l,m,p} \tag{6.12}$$

with $i = kn_{c\theta}+l$ and $j = mn_{c\theta}+p$. The sparsity is such that:

$$H_{4,k,l,m,p} \neq 0 \qquad \forall\, k+d_r \leq m \leq k-d_r, \forall\, l+d_\theta \leq p \leq l-d_\theta \tag{6.13}$$

Similarly H_2 and H_3 can be expressed using three indices (i,j,k) corresponding to the $\mathcal{C}^1$ basis spline, the radial 1D spline, and the poloidal 1D spline. The sparsity is such that:

$$H_{2,i,j,k} = H_{3,i,j,k} = 0 \qquad \forall\, j > d_r \tag{6.14}$$

6.2.1. Preconditioners

In this work the Jacobi preconditioner is used. This is quite a simple preconditioner and is therefore not expected to be the best preconditioner possible. Multiple different preconditioners were tested for this problem, however unfortunately none of those tested produced improved results. In this section we summarise the preconditioners that have been tested thus far.

In general these preconditioners will take the following form:

$$\begin{pmatrix} P_{H_1} & 0 \\ 0 & P_{H_4} \end{pmatrix} \tag{6.15}$$

where P_{H_1} and P_{H_4} are respectively the preconditioner for H_1 and H_4. P_{H_1} is typically either the inverse of the diagonal of H_1 or is simply the inverse of H_1 (easy to calculate as H_1 has only 9 elements).

Our aim is therefore to find a matrix which is sufficiently similar to H_4 while also being easily invertible.

6.2.1.1. Jacobi Preconditioner

This is the simplest preconditioner to implement. The preconditioner is the following:

$$\begin{pmatrix} \text{diag}(H_1)^{-1} & 0 \\ 0 & \text{diag}(H_4)^{-1} \end{pmatrix} \tag{6.16}$$

The use of this preconditioner approximately reduces the number of iterations by a factor 2.

6.2.1.2. FFT Preconditioner

This preconditioner is based on the idea that equidistant splines have the same shape. We therefore wish to approximate H_4 by a blockwise circulant matrix with circulant blocks. If the hypothesis is valid then we expect that this matrix will only differ from H_4 in the boundary regions. This preconditioner is based on work by Kormann and Sonnendrücker 2021. To increase the chances that the values on the diagonals of the matrix are the same this preconditioner will not approximate the problem matrix H_4 but rather a normalised version of this matrix:

$$\hat{H}_4 = \text{diag}(H_4)^{-\frac{1}{2}} H_4 \text{diag}(H_4)^{-\frac{1}{2}} \tag{6.17}$$

Once $\hat{H}_4$ has been constructed the preconditioner is constructed from one of its lines. The blockwise circulant matrix approximation with circulant blocks can be easily inverted using 2d Fourier transforms.

Experiments show that this method works effectively for the mass matrix but is ineffective when dealing with the stiffness matrix or a linear combination of the two matrices. This is presumably due to the fact that the hypothesis of the same values on the diagonals of the matrix is not valid (even after renormalisation using a Jacobi preconditioner) in non-cartesian geometry.

6.2.1.3. FFT Banded preconditioner

As the circulant hypothesis is invalid in the radial direction another possibility is to use the assumption in the poloidal direction only. This simplifies the problem such that instead of solving a matrix equation with $(n_{br}-2)n_{b\theta} \times (n_{br}-2)n_{b\theta}$ terms, we are able to solve $n_{b\theta}$ problems involving $(n_{br}-2) \times (n_{br}-2)$ sized banded matrices. Solving a reasonably sized banded matrix can be carried out easily using LAPACKAnderson, Bai, Bischof, et al. 1999. This solution works well for mass matrices and for the stiffness

matrix in the case of circular geometry. However for other arbitrary geometries, such as the ones that we will use in Sections 6.5 and 6.6, the metric tensor term $g_{s,\theta}$ is not necessarily 0. In this case this solution is no longer helpful.

6.2.1.4. Mass matrix as a preconditioner

Some papers suggest using the mass matrix as a preconditioner. This was tried briefly on a small test case but did not yield useful results. It also seems problematic as the mass matrix does not necessarily have an easily invertible form.

6.3. GmgPolar

In the design of the Geometric multigrid solver for curvilinear coordinates (GmgPolar) solver in Kühn, Kruse, and Rüde 2021; Martin J Kühn, Kruse, and Rüde 2022; Martin Joachim Kühn, Leleux, Kruse, et al. 2021, the authors focused on an interplay of a cheap discretisation technique with possible convergence to a higher order, and a fast solver for the solution of the linear system. A finite differences discretisation with the possibility for a matrix-free implementation was chosen, including an implicit extrapolation technique to increase the convergence order from 2 to 3 or 4. For the solution of the obtained linear system, a tailored multigrid (MG) method was developed. Multigrid methods exhibit low computational complexity, and can achieve high parallelism (Trottenberg, Oosterlee, and Schuller 2000). The family of geometric multigrid methods relies on mesh information, and is defined on a hierarchy of grids. Their design in the context of curvilinear coordinates was studied, e.g. in Barros 1988, and then generalised to curvilinear geometries in Martin J Kühn, Kruse, and Rüde 2022. In this section, we first briefly summarise the symmetric discretisation scheme for the Poisson-like equation, then we describe the corresponding geometric multigrid scheme introduced in Kühn, Kruse, and Rüde 2021; Martin J Kühn, Kruse, and Rüde 2022.

As in the case of the spline FEM solver, GmgPolar is defined on the domain represented by the curvilinear coordinates (r,θ). A standard 9-point finite differences discretisation of the partial differential equation (6.1) on the curvilinear domain would lead to a non-symmetric matrix. Since symmetric matrices are numerically advantageous, we instead discretise the energy functional

$$J(u) := \int_\Omega \left(\frac{1}{2}\alpha |\nabla u|^2 + \frac{1}{2}\beta u^2 - f u \right) d(x,y), \tag{6.18}$$

related to Equation (6.1) over a suitable Sobolev space incorporating the boundary conditions u_D. Here, $d(x,y)$ is the corresponding measure on Ω. This energy functional-focused approach maintains the symmetry of the matrix even for anisotropic grids, and yields a quadratic discretisation error. For more details, including the stencils in explicit form, see Kühn, Kruse, and Rüde 2021. In addition to this finite differences discretisation, we include a technique called implicit extrapolation. This technique

was introduced by Jung and Rüde in Jung and Rüde 1998 for a finite element discretisation on a hierarchical grid. In this approach, the system matrix is computed using a non-standard numerical integration rule and restructured, so that the obtained matrix is equivalent to the one obtained by the discretisation with quadratic finite elements. As a result, cubic convergence can be proven without having the extra cost from applying the numerical integration of the quadratic basis functions. In practice, we often even observe the convergence order 4. The application of that same idea to the finite differences scheme yielded similar results, see Martin J Kühn, Kruse, and Rüde 2022. For a more detailed motivation of the implicit extrapolation, see Schwarz 2021, Sec. 4.5 and the references therein. In both cases, with and without extrapolation, we obtain a matrix $A \in \mathbb{R}^{m \times m}$ with the size $m = n_r \cdot n_\theta$, where n_r is the number of nodes in the r-direction, and n_θ is the number of nodes in the θ-direction.

As additional requirement, the singularity at the origin of the mapping can also be handled. One option to circumvent this problem is the enforcement of Dirichlet boundary conditions on some small $r_0 > 0$. However, as this information is often synthetic and not available, another option is introduced. In Kühn, Kruse, and Rüde 2021, the heuristic discretisation approach "across the origin" was proposed. There, the origin is not chosen as a particular node of the mesh. Instead, the finite differences stencil for all points with (r_0, θ), $r_0 > 0$, is extended across the origin. In Martin J Kühn, Kruse, and Rüde 2022, it was shown that this approach yielded the same convergence order as with Dirichlet boundary conditions on the innermost circle if r_0 is reasonably small, e.g. $r_0 = 10^{-3}$.

As described above, a geometric multigrid method is applied to obtain an efficient solver with low memory requirements. As described in Section 6.1, the grid is obtained from a uniform refinement in each direction, and possible radial or poloidal additional refinement in order to take into account variations in the coefficient $\alpha(r)$ or the solution of Equation (6.1). One last uniform refinement is finally applied such that a node is added in the middle of all the intervals in r- and θ-directions. With this additional refinement, we obtain a locally structured grid, allowing a natural integration of the implicit extrapolation in the multigrid scheme (Jung and Rüde 1998; Martin J Kühn, Kruse, and Rüde 2022). We thus obtain a grid with n_r and n_θ nodes in the radial and poloidal directions respectively, i.e. in total there are again $m = n_r \cdot n_\theta$ nodes in the grid. To apply the multigrid scheme, we do not need one mesh but a set (or hierarchy) of meshes. Let us denote the finest level in the multigrid hierarchy of GmgPolar, the initial mesh introduced before, by Ω_1. Then, a hierarchy of $l > 1$ nested grid levels Ω_l are defined. These domains Ω_l are built using successive coarsening steps, i.e. such that $\Omega_{l+1} \subset \Omega_l$. We use standard coarsening by keeping points at $r = r_0$ and $r = a$, then taking one point over two in both directions of the polar plane.

The prolongation operator P_{l+1}^l which transfers the information between the consecutive grid levels $l+1$ and l is defined using bilinear interpolation for anisotropic meshes. The restriction operator from grid level l to $l+1$ is defined using the variational property $R_l^{l+1} = {P_{l+1}^l}^T$ (Briggs, Henson, and McCormick 2000).

For the problem of interest, but also for standard polar coordinates, standard coarsening combined with point smoothers is not efficient enough, partly due to the high

anisotropy of the problem represented with curvilinear coordinates. Hence, special care has to be taken to define each one of the multigrid components. One way to improve a multigrid solver is to use semi-coarsening in the direction of anisotropy (Trottenberg, Oosterlee, and Schuller 2000). Since in our approach we focus on standard coarsening techniques, the other possibility is to improve the smoothing procedure. Line smoothers such as circle (or radial) relaxation relax all the degrees of freedom (DOFs) of a circle (radius), i.e. DOFs with a constant radius r (angle θ). Note that we use the term circle or radius here although for deformed geometries one line does not represent a circle with constant radius. This means that we denote by a circle, a line of nodes (r,θ) with r constant.

Based on Barros 1988, it can be shown that the smoothing factors obtained with such relaxation schemes highly depend on the position in the domain. In particular, a circle line smoother is efficient on the interior of the domain, and a radial line smoother is efficient on the exterior part. This is explained by the fact that polar (or certain curvilinear) transformations imply strong connections between DOFs on circle lines on the interior part of the circular domain, and strong connections between DOFs on radial lines on the outer part. To address this problem, the type of smoother is switched from circle to radial for nodes where $\frac{k_j}{h_i} r_i > 1$ with r_i the radius, h_i the next radial interval, and k_i the next poloidal interval. This is a simple heuristic obtained from the analytical expression of the smoothing factor in Barros 1988, and has been empirically shown to be the best choice also in our case, see Martin J Kühn, Kruse, and Rüde 2022. In the implementation of GmgPolar, we thus partition the domain into two subdomains, corresponding to each smoother coloured alternatingly in black and white. When using compact stencils for each smoother, as it is the case here, all lines with the same colour are then independent, see Figure 6.3. This is useful to obtain a partial parallelisation of the combined relaxation method.

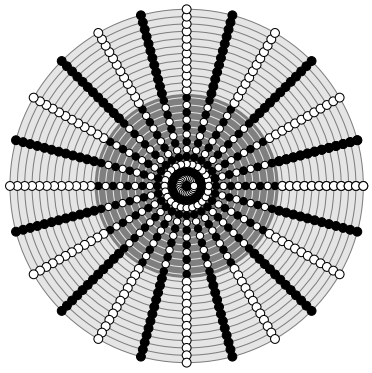


Figure 6.3.: The circular domain is split in two subdomains (shown with light grey and dark grey background colours) depending on the use of a circle line smoother (dark grey) and a radial line smoothing (light grey). The nodes on these subdomains are coloured alternatingly in black and white. Corresponding colouring schemes are used for deformed geometries such as Figure 6.6a or Figure 6.6b.

The resulting smoother is an alternating zebra relaxation consisting in the successive application of the circle and radial smoothers, similar to a block Gauss-Seidel method, with a parallel handling of black and white coloured lines. Note that the two subdomains could also be smoothed in parallel, similarly to a block Jacobi method. However this leads to slightly worse iteration numbers, and a similar speed-up can be obtained with partial parallelisation, see Martin J Kühn, Kruse, and Rüde 2022. To apply the block Gauss-Seidel type of smoother, small linear systems must be factorised (once) and solved, each corresponding to one circle or radial line, and this can also be performed in parallel. When using the "across-the-origin" discretisation, the linear system solved in the smoother for the first circle of the polar plane produces a large fill-in upon factorisation, which can affect the performance of the solver. In order to mitigate this downside, we handle the corresponding system using the state-of-the-art sparse direct solver MUMPS[1] (Amestoy, Duff, L'Excellent, et al. 2001), with the version 5.4.1.

Based on all of the previous elements, the multigrid scheme, possibly combined with an implicit extrapolation (Jung and Rüde 1998; Martin J Kühn, Kruse, and Rüde 2022) that only needs to act between the two finest grids, uses a traditional V-cycle. The observed convergence order when using implicit extrapolation is up to 4, see Martin J Kühn, Kruse, and Rüde 2022. The asymptotic complexity of GmgPolar can be shown to be optimal, i.e. linear with respect to the size of the matrix, in the sense that

- the convergence of the multigrid scheme is mesh-independent as shown empirically in Martin J Kühn, Kruse, and Rüde 2022.
- the computational and memory complexities are linear, except for the cost of the coarsest grid correction which becomes negligible when enough levels are used in the multigrid hierarchy. This is shown in Martin Joachim Kühn, Leleux, Kruse, et al. 2021. We use the direct solver MUMPS to solve the system on the coarsest level.

The main implementation of the GmgPolar solver follows a matrix-free scheme, i.e. the matrices are not assembled but constructed and applied on-the-fly. The solver also include the possibility to use matrices which are fully assembled during the initialisation phase.

6.4. Embedded boundary solver based on AMReX

An alternative possibility to handle partial differential equations in complex geometries is to use a simple Cartesian mesh, and to cut out the computational domain based on a level set function defining the interior, boundary, and exterior of the domain. Compared to the use of a curvilinear mesh, this approach yields a simpler structure and operations related to the coordinate transformation are avoided. Most importantly however, there is no need for a single coordinate transformation, so more

[1] http://mumps.enseeiht.fr/

complex geometries—like an X-point geometry—can be handled in the same way. Several approaches of such embedded boundary methods have been proposed in recent years, including approaches based on finite volumes (Johansen and Colella 1998; Berger and Helzel 2012), the cut finite elements method (Burman, Claus, Hansbo, et al. 2015), and the finite cell method (Parvizian, Düster, and Rank 2007). The AMReX library (AMReX Development Team, A. Almgren, Beckner, et al. 2022; al. 2019) implements a finite volume solver based on the work of Johansen and Colella 1998. The discretisation is based on a box that includes the full physical domain. Then, the physical domain is cut out of the box by finding the intersections of the domain boundary and the cell boundaries. The physical domain is finally represented by the piecewise linear representation connecting the intersection points with the cell boundaries. Figure 6.4 illustrates the situation.

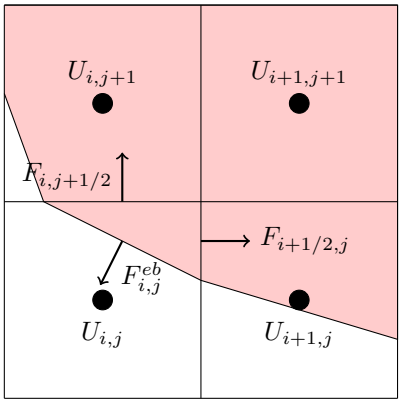


Figure 6.4.: Illustration of the finite volume approach with embedded boundaries. The shaded part of the cells is inside, the white part outside of the domain. The fluxes through the boundary of the lower left cell are shown.

For the finite volume solution, Equation (6.1) is reformulated as follows by integrating over one cell. Let $U_{i,j} \approx u((i-1/2)\Delta x, (j-1/2)\Delta y)$ be the value in the middle of the cell (i,j) of the Cartesian grid. Then, the differential operator L in Equation (6.1) is discretised as

$$(LU)_{i,j} = -\frac{1}{\Delta x \Delta y}\left(F_{i+1/2,j} - F_{i-1/2,j} + F_{i,j+1/2} - F_{i,j-1/2}\right) + \beta(x_i, y_j)U_{i,j}, \tag{6.19}$$

where Δx, Δy are the length of the cell in x and y, respectively, and the fluxes $F_{i+1/2,j}$ at the cell boundary point $(x_i + \frac{\Delta x}{2}, y_j)$ are given as

$$F_{i+1/2,j} = \Delta y\, \alpha(x_{i+1/2}, y_j)\frac{U_{i+1,j} - U_{i,j}}{\Delta x}. \tag{6.20}$$

For cells that are cut by the boundary, this formula has to be modified to account for the volume fraction $\Lambda_{i,j}$ included in the physical domain, the area fraction $a_{i+1/2,j}$ of the face $(i+1/2, j)$, called aperture, as well as the addition face defined by the line

connecting the intersection points of the cell boundaries and the physical boundaries. The approximation of the operator in Equation (6.19) is modified on the partially covered cells to

$$(LU)_{i,j} = -\frac{1}{\Delta x \Delta y \Lambda_{i,j}} \left(F_{i+1/2,j} - F_{i-1/2,j} + F_{i,j+1/2} - F_{i,j-1/2} - F^{eb}_{i,j} \right) + \beta(x_i, y_j) U_{i,j}. \tag{6.21}$$

For the fraction on the cell boundary, the modified flux formula is given as

$$F_{i+1/2,j} = a_{i+1/2,j} \Delta y \alpha_m \left(\frac{1 + a_{i+1/2,j}}{2} \frac{U_{i+1,j} - U_{i,j}}{\Delta x} + \frac{1 - a_{i+1/2,j}}{2} \frac{U_{i+1,j+1} - U_{i,j+1}}{\Delta x} \right), \tag{6.22}$$

where α_m is a linear interpolation of the value of α at the midpoint of the partial cell boundary. The flux through the boundary $F^{eb}_{i,j}$ is computed from the value at the boundary and a linearly interpolated value along the first intersection of the inward-pointing normal and a cell-boundary. The fluxes are illustrated in Figure 6.4. Note that this approximation of the flux is only first order accurate, while the rest of the method is second order accurate. Johansen and Colella 1998 have proposed a second order reconstruction of the flow through the boundary. However, we use the first order version, as we rely on AMReX which only implements this version. The complete solver is observed to have second order despite the reduced order on the 1D curve. The resulting system is then solved based on a matrix-free geometric multigrid solver with a Gauss–Seidel smoother and a biconjugate gradient stabilised coarse grid solver. The solver can handle any number of points, however, a ratio of 2 between various levels is fixed. On a uniformly refined grid, let the number of cells per direction be given as $n = 2^l m$, where m is not divisible by 2. This means, the number of levels in the multigrid solver is restricted to l at most and the coarse grid solver has to solve a system with at least m points in this direction. If the number m becomes too large, the convergence of the coarse grid solver can become slow and the complete solver is inefficient. Patches containing multiple grid cells can be further refined where a maximum refinement ratio of 1:4 is enforced on boundaries of different levels of refinement. The reflux coarse-fine boundary update that is implemented in AMReX to enable multigrid solution with refined patches is described in A. S. Almgren, J. B. Bell, Colella, et al. 1998.

An important ingredient in the construction of the problem is the definition of the level set function to find the intersection of the physical boundary with the cells of the computational grid. For the mappings considered in this chapter, the boundary corresponds to a level set $r = a$ of a radial mapping of the form

$$(x, y) = F(r, \theta).$$

Both the coefficients α and β in Equation (6.1) and the right-hand-side are given as functions of (r, θ), which is the usual case in the context of the GYSELA code that uses the curvilinear coordinate system. In order to evaluate the functions at the grid point and in order to reconstruct the physical boundary, we thus need to invert this

mapping. In simple cases the inversion can be found analytically, but in general this needs to be obtained numerically. This is done with a Newton iteration starting from a good initial guess. As an initial guess, it turns out that a circular mapping around the singular point $F(0,0)$ is a good guess in a small circle around the singular point. Further out, we evaluate the mapping on a fine regular grid in (r,θ) and use the closest point on this grid as a starting guess for the Newton iteration.

6.5. Comparison of the three solvers for the gyrokinetic Poisson-like equation

We now compare the three solvers on analytical test cases. Our goal is to show the advantages of each solver and estimate

1. in which conditions each solver should be preferred for use in the GYSELA code,

2. how far is each solver from meeting the requirements of realistic plasma simulations.

The GmgPolar solver follows a matrix-free implementation with two different schemes, with and without implicit extrapolation (for more details, see Section 6.3). In our analysis, we consider both cases. In the case of the GmgPolar solver with implicit extrapolation, we also include performance results for an implementation using fully assembled matrices, which are stored in memory. Similarly for the Spline FEM solver, the degree is a parameter of the method and any value can be used. We consider two configurations, specifically quadratic and cubic splines. Cubic splines are chosen as the splines used in GYSELA are also cubic (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016). Quadratic splines are chosen to provide a comparison case with an order closer to that of the other schemes studied. A larger degree could also have been used to obtain a higher convergence rate. The accuracy order is not configurable in the current version of the Embedded Boundary solver.

The maximum of the residual is used as the stopping criteria for the iterative methods used by each solver to solve the linear system that they describe.

6.5.1. Test cases

The solvers will be compared based on several criteria. We expect that different solvers will be better adapted to different geometries or solutions. Two different analytical geometries and three different manufactured solutions are therefore used for this study. In addition we provide analytical definitions of the coefficients α and β in Equation (6.1). In curvilinear coordinates, these coefficients depend only on the

radius r. We define the coefficients similarly to Zoni 2019:

$$\alpha(r) = \exp\left[-\tanh\left(\frac{r - r_p}{\delta_r}\right)\right], \tag{6.23}$$

$$\beta(r) = -1/\alpha(r). \tag{6.24}$$

In contrast to their approach, we consider a steeper gradient $\delta_r = 0.05$, nearer to the wall $r_p = 0.7$. This situation is slightly more realistic in the case of a tokamak plasma. The radial profile of the diffusivity coefficient $\alpha(r)$ can be seen in Figure 6.5.

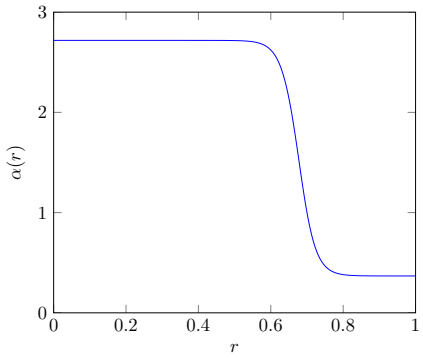


Figure 6.5.: Radial profile of the diffusivity coefficient $\alpha(r)$ defined in (6.23) for the gyrokinetic Equation (6.1).

The first geometry is a stretched ellipse with a Shafranov shift defined by the mapping

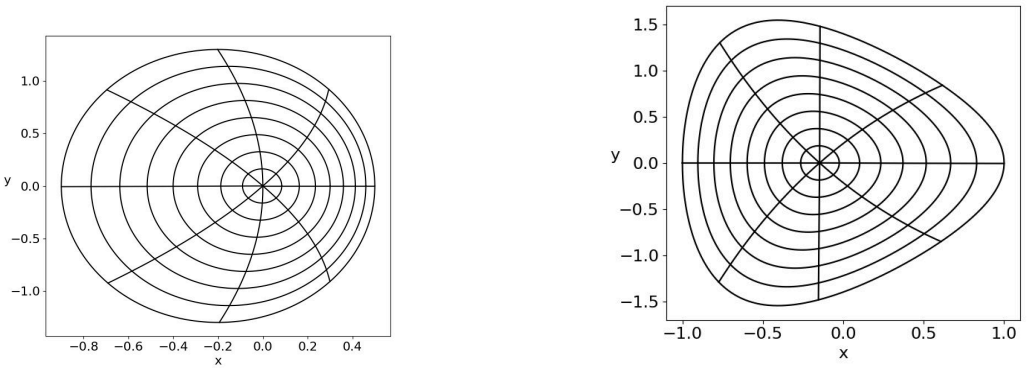
$$\begin{aligned} x(r,\theta) &= (1 - E_0)\, r\cos\theta - \delta_0 r^2, \\ y(r,\theta) &= (1 + E_0)\, r\sin\theta, \end{aligned} \tag{6.25}$$

where E_0 is the elongation and δ_0 the Shafranov shift. In our investigations, we refer to this geometry as the "Shafranov" geometry, and use the same parameters as Zoni and Güçlü 2019: $E_0 = 0.3$, and $\delta_0 = 0.2$. The second geometry, originally proposed by Czarny and Huysmans 2008, has a triangular shape and ellipticity, and is defined by the mapping:

$$\begin{aligned} x(r,\theta) &= \frac{1}{\varepsilon}\left(1 - \sqrt{1 + \varepsilon\,(\varepsilon + 2r\cos\theta)}\right), \\ y(r,\theta) &= y_0 + \frac{e\xi\, r\sin\theta}{2 - \sqrt{1 + \varepsilon\,(\varepsilon + 2r\cos\theta)}} = y_0 + \frac{e\xi\, r\sin\theta}{1 + \varepsilon x(r,\theta)}, \end{aligned} \tag{6.26}$$

where y_0 corresponds to the centre of the mapping, ε is the inverse aspect ratio, e the ellipticity, and $\xi = 1/\sqrt{1 - \varepsilon^2/4}$. In our investigations, we refer to this geometry as the "Czarny" geometry, we use the same parameters as Zoni and Güçlü 2019: $y_0 = 0$, $\varepsilon = 0.3$ and $e = 1.4$. Figure 6.6 shows the "Shafranov" and the "Czarny" geometries.

These geometries have previously been investigated by Bouzat, Bressan, Virginie Grandgirard, et al. 2018 and Zoni and Güçlü 2019.



(a) "Shafranov" geometry defined by Equation (6.25) with $E_0 = 0.3$ and $\delta_0 = 0.2$

(b) "Czarny" geometry defined by Equation (6.26) with $y_0 = 0$, $\varepsilon = 0.3$, and $e = 1.4$

Figure 6.6.: Analytical geometries of the domain for the 2D gyrokinetic Poisson-like equation.

We now introduce three manufactured solutions which respect the homogeneous Dirichlet boundary conditions:

1. **Polar solution:** A solution with oscillations aligned with the polar grid which can be used as an initial perturbation in the GYSELA code (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016):

$$u(x,y) = C(r(x,y))^6 (r(x,y)-1)^6 \cos(m\theta), \tag{6.27}$$

where $r(x,y)$ is the radial coordinate defined by the mapping, $C = 2^{12} \cdot 10^{-4}$ and $m = 11$.

2. **Cartesian solution:** A solution with oscillations aligned with the Cartesian grid:

$$u(x,y) = C\left(1 + r(x,y)\right)^6 \left(1 - r(x,y)\right)^6 \cos(2\pi x) \sin\left(2\pi y\right), \tag{6.28}$$

where $r(x,y)$ is the radial coordinate defined by the mapping, and $C = 2^{12} \cdot 10^{-4}$.

3. **Multi-scale solution:** A solution with large oscillations aligned with the polar grid in the centre, and small oscillations, also aligned with the polar grid, near the edge region. This solution mimics the physics in a tokamak where large structures appear near the centre, and smaller structures appear near the edge (Guilhem Dif-Pradalier, Philippe Ghendrih, Yanick Sarazin, et al. 2022). The solution is defined as:

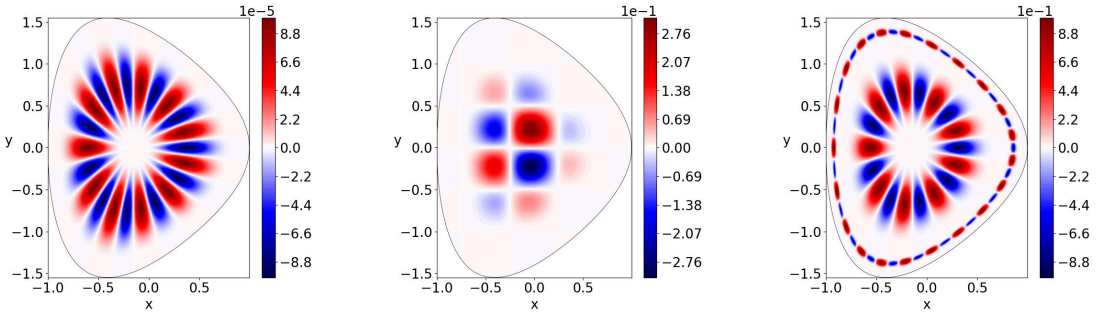
$$u(x,y) = h(r(x,y),\, 0.45,\, 0.02)\cos(9\theta) + h(r(x,y),\, 0.9,\, 0.0003)\cos(21\theta), \tag{6.29}$$

where $h(r,c,w)$ is a function, constructed from the Gaussian functions $g(r,c,w)$ centred on c with standard deviation w. $u(x,y)$ is constructed such that its value and its first derivative are continuous at the singular point:

$$h(r,c,w) = g(r,c,w) - r\frac{\partial g}{\partial r}(0,c,w) - g(0,c,w) + \left(r\frac{\partial g}{\partial r}(0,c,w) + g(0,c,w) - g(1,c,w)\right) r^2, \tag{6.30}$$

$$g(r,c,w) = \exp\left(-(r-c)^2/w\right), \tag{6.31}$$

The right hand side f corresponding to these solutions is obtained analytically. Figure 6.7 shows the shape of all three solutions on the "Czarny" geometry defined in Equation (6.26).



(a) Polar solution defined in Equation (6.27)

(b) Cartesian solution defined in Equation (6.28)

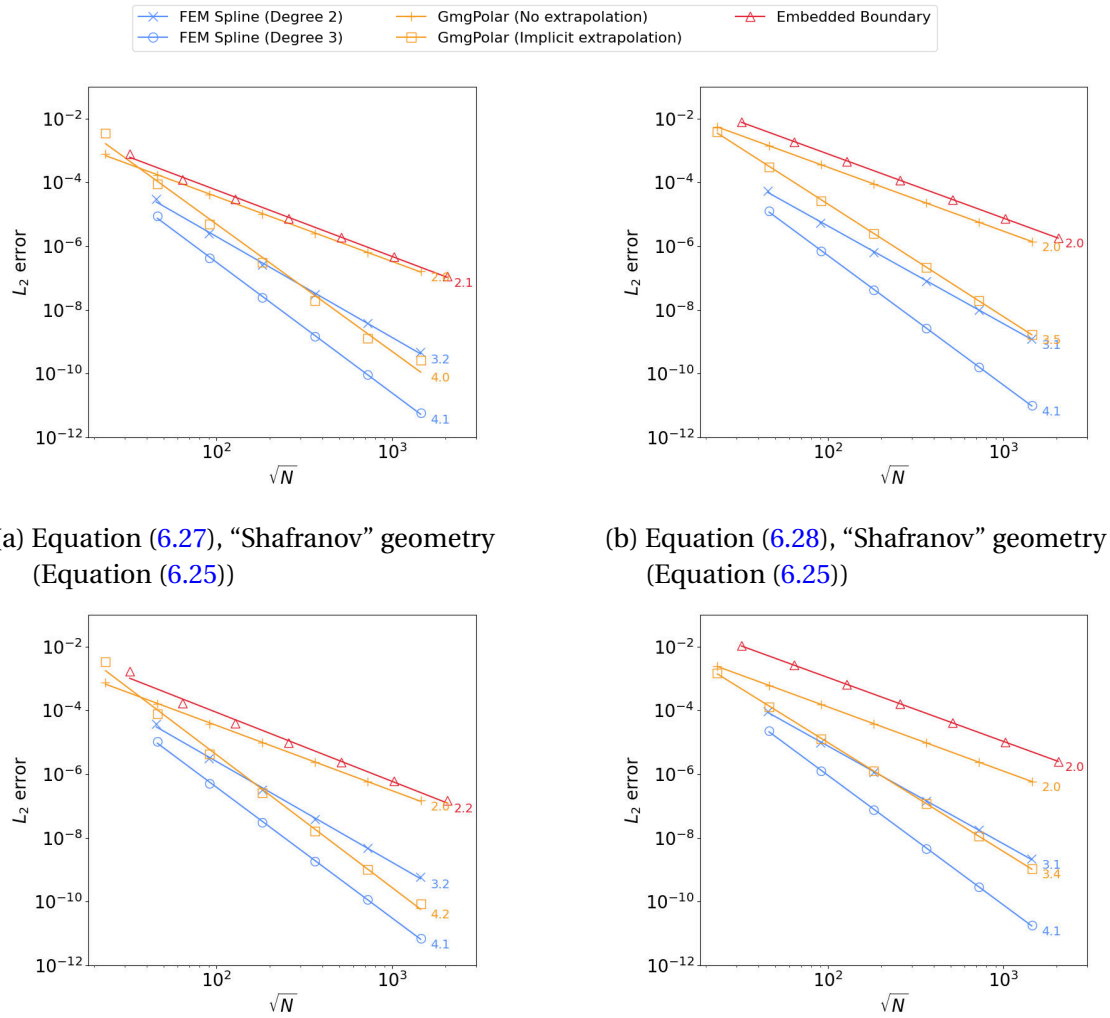
(c) Multi-scale solution defined in Equation (6.29)

Figure 6.7.: Shape of the manufactured solutions on the "Czarny" geometry defined by Equation (6.26).

We begin by considering four test cases on equidistant meshes: the polar solution (Figure 6.7a) and the Cartesian solution (Figure 6.7b) defined on both the "Shafranov" geometry (Figure 6.6a) and the "Czarny" geometry (Figure 6.6b). Following this study, we will focus on the "Czarny" geometry, which exhibits stronger poloidal anisotropy, to examine the effects of local grid refinement for the previous solutions as well as the multi-scale solution (Figure 6.7c).

6.5.2. Accuracy

The results of the L_2-error convergence from the application of the three solvers for the four equidistant cases can be seen in Figure 6.8. In order to ensure that each solver has converged to the most accurate result possible, the stopping criteria is set to 10^{-14} for the Spline FEM solver, and 10^{-11} for the other solvers. The errors are plotted as a function of $\sqrt{N}$, where N is the total number of points in the simulation ($N = N_r N_\theta$ for the Spline FEM solver and the GmgPolar solver, $N = N_x N_y$ for the Embedded



(a) Equation (6.27), "Shafranov" geometry (Equation (6.25))

(b) Equation (6.28), "Shafranov" geometry (Equation (6.25))

(c) Equation (6.27), "Czarny" geometry (Equation (6.26))

(d) Equation (6.28), "Czarny" geometry (Equation (6.26))

Figure 6.8.: L_2 error normalised by the ∞-norm of the solution as a function of the total number of points N, when solving different equations on different geometries with the five solver configurations.

Boundary solver), which ensures that the gradient is equal to the order of convergence. The expected orders of convergence are obtained for all solvers. The Spline FEM solver converges with an order equal to $d+1$, where d is the degree of the spline. The GmgPolar solver without extrapolation has second order convergence, while the extrapolation increases the order to 4 for the polar solution, and around 3.5 for the Cartesian solution. The Embedded Boundary solver has second order convergence. A larger error is observed for all solvers in the case of the Cartesian solution, defined by Equation (6.28), even the Embedded Boundary solver despite the fact that it is based on a Cartesian grid. This is because the coefficients $\alpha(r)$ and $\beta(r)$ are still defined radially. Thus, the solution is not entirely aligned with a Cartesian grid, but also contains a radial component. We see that the cubic Spline FEM solver outperforms the other solvers in terms of L_2 error. The GmgPolar solver with implicit extrapolation provides a similar accuracy to the quadratic Spline FEM solver, outperforming it for the cases with 512 or more cells.

The choice of the geometry does not seem to influence the accuracy of the solvers, and it has no effect on the operations performed. As a result, in what follows, we focus on a single geometry. Specifically, the "Czarny" geometry is used to allow the investigation of poloidal anisotropy.

The smaller error in the spline case allows fewer points to be used to attain the same precision as other methods. A smaller number of points can reduce the memory requirements and decrease the execution speed. On the other hand, different resolution methods have different memory requirements, so the fact that a method requires the lowest number of points is no guarantee that it will demonstrate the best performance.

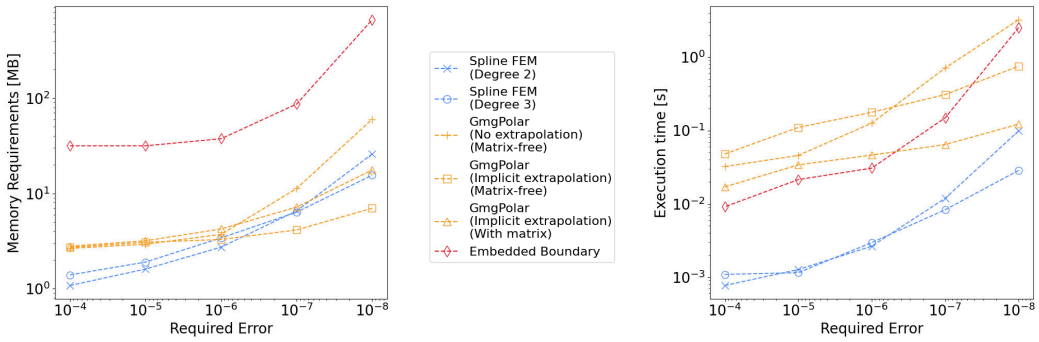
6.5.3. Performance

In this section, we compare the memory consumption and execution times of the three solvers. First, we target a fixed error to be attained, using the lowest number of points for each solver, to estimate their behaviour in an actual simulation code. Then, we use fixed problem sizes in order to estimate the computational efficiency of each solver. Finally, the parallel scalability is detailed for all solvers for a problem of size $4 \cdot 10^6$. The stopping criteria is set to 10^{-8} for all solvers. The tests were run at the Centre de Calcul Intensif d'Aix Marseille. The cluster uses Intel Xeon Gold 6142 (Sky Lake) cores at 2.6 GHz, for a theoretical peak performance of 579 TFLOPS/s. Each of its 158 compute nodes is a 2-socket system with 192 GB memory, where the 16 cores of each socket constitute a separate NUMA (non-uniform memory access) domain. The cluster uses the Intel OmniPath interconnect. All three codes are compiled with the "-O3" flag, but no further compiler-based optimisations are applied. Execution times are calculated by taking the average time measured over ten runs.

In the case of the GmgPolar solver, the main implementation follows a matrix-free scheme, i.e. the matrices required for the multigrid iterations are directly applied on-the-fly and never stored. In order to highlight the specificities from the matrix-free implementation, denoted by "matrix-free", we also include in the following, the results from an implementation of the GmgPolar solver with extrapolation where the matrices

are assembled and stored in memory during the initialisation phase, denoted by "with matrix".

In order to investigate the performance, in terms of computation and memory cost as a function of a required error, we must first determine the minimum number of points required to obtain an error lower than the required error. This is achieved via a binary search (Knuth 1998). In order to handle the two dimensions the binary search is carried out while ensuring that there are the same number of points in both dimensions. Once the smallest number of points respecting this criteria has been found, a second binary search is carried out along whichever dimension can use fewer points without the error passing the limit. For the GmgPolar and spline FEM solvers this is the radial dimension as, due to the curvilinear coordinates, significantly more points (around a factor of 4) are needed in the θ-direction to correctly model the solution shown in Figure 6.7a. The Embedded Boundary solver solution is equally constrained in both dimensions. Experiments have shown that decreasing the number of points in either dimension leads to the error being exceeded. Both GmgPolar and the Embedded Boundary solver rely on multigrid methods. To ensure that the required hierarchy of grids can be constructed using standard coarsening, the number of points in each direction is chosen such that it can be expressed as $N_C \cdot C2^{l-1}$, where l is the chosen number of levels, N_C is the minimum number of points in each direction on the coarsest grid, generally $N_C = 2$, and $C \leq 5$ is not divisible by two.



(a) Memory requirements

(b) Serial execution speed

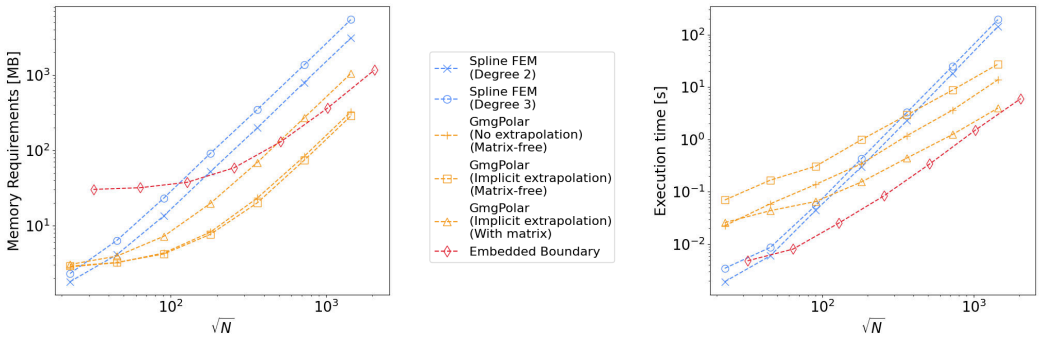
Figure 6.9.: Performance in terms of a) memory requirements and b) CPU time, as a function of the desired error, for the solution described by Equation (6.27) on the "Czarny" geometry described by Equation (6.26).

Figure 6.9 shows the serial performance of the different methods as a function of the required error, for the polar solution, described by Equation (6.27), on the "Czarny" geometry, described by Equation (6.26). Figure 6.9a gives the memory consumption for all solvers, with respect to the required error. For target errors smaller than 10^{-7}, the GmgPolar solver with implicit extrapolation has the lowest memory requirements, followed closely by the Spline FEM solver requiring around 2 times more memory for the best accuracy, and finally the Embedded Boundary solver consuming 100 times

more memory. The low consumption of the GmgPolar solver is mainly due to the matrix-free implementation in which the matrix operators are constructed on-the-fly instead of being stored explicitly, as well as the high order of the method. Although the Embedded Boundary solver is also matrix-free, its definition of the boundary consumes a large amount of memory. When constructing the matrix in the GmgPolar solver explicitly, the memory consumption grows 2 times larger, close to the cubic Spline FEM solver. In the case of the GmgPolar solver without the extrapolation, the lower order of the method implies the use of more points to attain the same accuracy, and thus more memory. Figure 6.9a also shows that the Embedded Boundary solver has a minimum memory usage of around 20MB. This is due to optimisations inside the AMReX library which allocates memory in batches in order to optimise the memory management. Similarly, the GmgPolar solver has a minimum memory usage of around 3MB, which corresponds to the initial allocation of the MUMPS library to handle both the coarsest problem and the singularity of the polar plane in the smoother. The Spline FEM solver is a high order method requiring a smaller number of points to attain the same accuracy, and thus uses less memory than the GmgPolar solver for target errors larger than 10^{-7}.

The serial execution time is also compared. In order to provide the most pertinent information for an implementation in the GYSELA code where the solver will be executed multiple times without modifying the setup, the initialisation phase is excluded from the execution time. In the execution time comparison, shown in Figure 6.9b, the smaller number of points required to attain a fixed error leads to the spline FEM solver showing the best performance. For a required error of 10^{-8}, the GmgPolar solver with extrapolation is approximately 25 times slower, and the Embedded Boundary solver is 90 times slower than the cubic Spline FEM solver. In the case of the GmgPolar solver, the slower execution time is also due to the overhead from the matrix-free implementation: while we do not need to store the matrix operators of the multigrid scheme, they must be reconstructed on-the-fly at each step of each iteration, which is expensive. GmgPolar with extrapolation can be sped up significantly (around 4 times faster) when using the assembled matrix version, denoted by "with matrix". The Embedded Boundary solver is slightly faster than the GmgPolar solver for target errors larger than 10^{-7} despite requiring at least three times as many points for these cases. This is due to the fact that the code relies on the heavily optimised AMReX library(Zhang, A. Almgren, Beckner, et al. 2019). It is important to note that this speed comparison based on error requirements determined via binary searches is not the most advantageous for GmgPolar and the Embedded Boundary solver. Both use multigrid methods to solve the equations which restricts the choice of points found with the binary search to a multiple of powers of 2 such that an efficient multigrid preconditioning can be obtained with the current implementations. This number of points may then be significantly larger than necessary to attain a required error, compared to the Spline FEM solver which can use the minimum number of points.

In the context of a plasma simulation such as the GYSELA code (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016), the error arising from the Poisson solver may not be the limiting factor for the choice of points. Other methods used in the simulation,



(a) Memory requirements

(b) Serial execution speed

Figure 6.10.: Performance in terms of a) memory requirements and b) CPU time, as a function of the total number of points N, for the solution described by Equation (6.27) on the "Czarny" geometry described by Equation (6.26).

such as the advection or the collision operators, may be less accurate and require more points than would be necessary for simply solving the quasi-neutrality equation. It is therefore equally important to examine the performance as a function of the number of points. Figure 6.10 shows the serial performance of the different methods for the solution described by Equation (6.27) on the "Czarny" geometry described by Equation (6.26), as a function of the problem size. The results for the Embedded Boundary solver are obtained with $n_x = n_y$, where n_x and n_y are the number of points in the x and y directions, while the results for the GmgPolar solver and the Spline FEM solver are obtained with $2n_r = n_\theta$, where n_r and n_θ are the number of points in the r and θ directions. The ratio $2n_r = n_\theta$ is chosen to match the usual ratio used in GYSELA (imposed by physical considerations). All solvers have a memory consumption growing linearly as the number of points increases, making the memory requirements easy to predict for larger cases. We see that the GmgPolar solver with the implicit extrapolation has the lowest memory consumption for problems of size 10^3 or larger. This memory describes around 10 vectors of size N. On a single node of the same cluster, with 192GB memory, it would then be possible to solve a problem of size 10^9 using the matrix-free GmgPolar solver, of size 10^7 for the cubic Spline FEM solver, and of size 10^8 for the other solvers. In the case of the GmgPolar solver, using the implicit extrapolation does not increase memory consumption, as the solver still benefits from the matrix-free implementation. We also provide the requirements for the assembled matrix version which grow similarly but needs about five times more memory than the matrix-free version. The memory requirements for the Embedded Boundary solver are also reasonable, especially for larger cases, with only 3 times more memory than the GmgPolar solver for the largest size. In contrast, the Spline FEM solver is very memory heavy, requiring approximately ten times more memory than the GmgPolar solver. Again, we observe that both the Embedded Boundary solver and the GmgPolar solver have a minimum memory usage with a plateau for small problem

sizes.

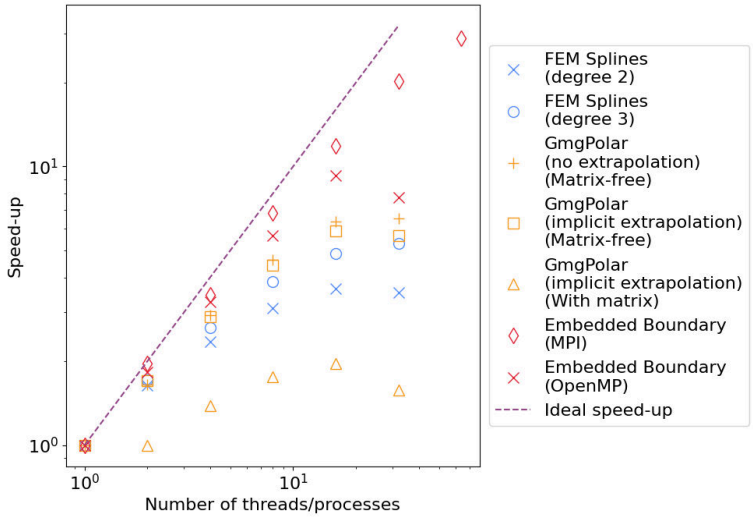


Figure 6.11.: Execution time to solve for the polar solution described by Equation (6.27) on the "Czarny" geometry described by Equation (6.26) on 2048×2048 points.

Figure 6.10b shows the serial execution time as a function of the total number of points N. We see that the Embedded Boundary solver is the fastest, followed by the GmgPolar solver for problems larger than 10^5. Using the assembled matrices in GmgPolar improves the execution times by a factor of seven for the largest problems, making it less than two times slower than the Embedded Boundary solver. The Spline FEM solver takes longer to solve larger problems, with the execution time increasing faster than the other two solvers. The execution times shown for the GmgPolar solver illustrate the advantage that can be gained by using the largest possible number of levels.

We now have highlighted the serial performance of each solver in terms of memory consumption and execution time. However, an effective parallelisation is also a crucial point for the simulation of large systems. The GmgPolar solver and the Spline FEM solver use OpenMP to accelerate the code. The Embedded Boundary solver has the advantage of being based on the parallel library AMReX(Zhang, A. Almgren, Beckner, et al. 2019). As a result, it can be run with both OpenMP and MPI. Please note that the matrix-version of GmgPolar has not been well parallelised so far, as the focus has been the matrix-free version to reduced the memory consumption. It is only printed for completeness. Figure 6.11 shows the performance of each solver in a parallel setup for the polar solution described by Equation (6.27) on the "Czarny" geometry described by Equation (6.26) using 2048×2048 points. Up to 16 threads, the OpenMP parallelisation of all three solvers show similar speed-ups, then we have a speed-down when arriving at 32 threads. This speed-down is due to the threads being placed in separate sockets inside the node when using all 32 threads, which slows

down the communication between threads, and thus increases the overall execution time. We see that the Embedded Boundary solver has the most efficient parallelisation especially when MPI is used, where the speed-up continues to grow further with the use of 32 processes. This MPI parallelism, makes it the only solver capable of using more than one node, or efficiently exploiting all 32 threads inside a node.

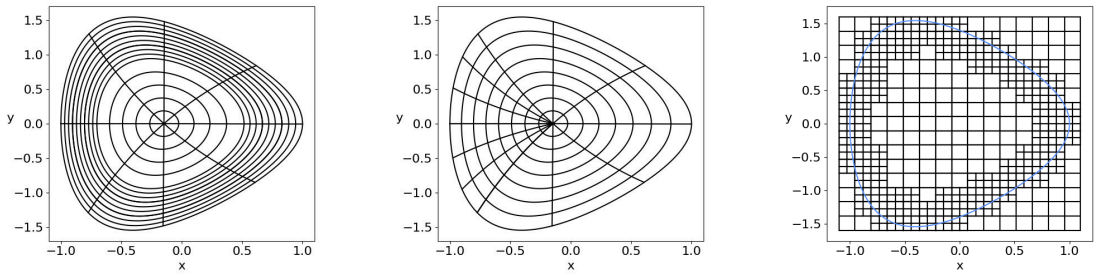
To summarise, for equidistant points Figure 6.8 shows that the Spline FEM solver allows us to obtain the smallest errors, and therefore the smallest number of points for a given error. Thus, Figure 6.9b shows that the Spline FEM solver is the fastest to attain a specific error. Then, the results from Figure 6.10b show that the Embedded Boundary solver is the fastest in terms of degrees of freedom per second, with the matrix-version of GmgPolar coming close for the largest problems considered. Additionally, the Embedded Boundary solver has the best parallelisation, as illustrated in Figure 6.11. Finally, the results from Figures 6.9a and 6.10a show that the GmgPolar solver has the lowest memory requirements when using the matrix-free implementation and, when used with implicit extrapolation, seems to present a good compromise between a solution obtained with relatively fast speed, and a high order approximation involving a small number of points.

6.5.4. Refinement

Having shown that the three methods work as expected for uniform points, we will now tackle the more complex case of non-uniform points. In the following, we use three different ways to refine the domain:

1. around a certain radius in order to accurately capture the variations of the coefficient $\alpha(r)$, or the variations of the solution (as seen in the multi-scale solution shown in Figure 6.7c), i.e. a localised radius refinement as shown in Figure 6.12a.

2. in the θ direction in order to account for the anisotropy introduced by the curvilinear coordinates, i.e. a localised θ refinement, as shown in Figure 6.12b.



(a) Radial refinement

(b) Poloidal Refinement

(c) Refinement on Cartesian patches

Figure 6.12.: Different ways to define and refine grids on the Czarny geometry.

3. on a specific patch where the error is higher, as shown in Figure 6.12c.

While these additional refinements allow specific effects in the domain to be treated with a greater accuracy, they also require very specific developments for the discretisation and the solver. As detailed in Sections 6.2 and 6.3, the Spline FEM solver and the GmgPolar solver can be refined in the r-direction or the θ-direction. The Embedded Boundary solver can refine on patches as described in Section 6.4.

As in Section 6.5.2, the stopping criteria is set to 10^{-14} for the Spline FEM solver, and 10^{-11} for the other solvers.

Figure 6.13 shows the errors obtained when solving for Equation (6.28) on the "Czarny" geometry defined by Equation (6.26) with 64×64 points. This geometry demonstrates anisotropy due to the triangular shape. Although the small number of points used in this test case allows a reasonable approximation of the solution to be obtained (L_∞ norm of the error is $3.51 \cdot 10^{-5}$ for the Spline FEM solver, $8.29 \cdot 10^{-4}$ for the GmgPolar solver, and $7.27 \cdot 10^{-3}$ for the Embedded Boundary solver), it is not sufficient to adequately resolve the equation poloidally. This highlights the anisotropy problems. The Embedded Boundary solver is unaffected by this as it does not use the geometry to define the grid, hence the error in Figure 6.13c has a similar shape to that of the manufactured solution shown in Figure 6.7b. In contrast, the spline FEM and GmgPolar solvers show increased errors in the negative x region where the poloidal points are more widely spaced. Poloidal refinement can be used to counteract this effect. Figure 6.14 shows how the error is affected by adding 50% more points in the region $\theta = [-\pi/4, \pi/4]$. We see that this change is more than sufficient to compensate the additional error due to the anisotropy for both solvers.

In a tokamak plasma simulation, radial refinement is often desired as the turbulence is created at larger scales in the central region compared to the edge region (Guilhem Dif-Pradalier, Philippe Ghendrih, Yanick Sarazin, et al. 2022). In order to examine this possibility, we will investigate the solution shown in Figure 6.7c and described by Equation (6.29), which is designed to mimic this situation. The GmgPolar and spline FEM solvers will use radial refinement in the region $r \in [0.8, 1.0]$ to handle the steeper gradients seen in u and f in this region, as shown in Figure 6.12a. The Embedded Boundary solver will use patches overlapping this region as shown in Figure 6.12c. Three cases will be compared, the first is the uniform case with no refinement. In this case, the Embedded Boundary solver will use the same number of points in the x and y directions. The Spline FEM solver and the GmgPolar solver will use four times as many points poloidally as radially. In the second case, denoted by "1 refinement", the number of points is doubled in the region of interest. The GmgPolar solver needs the total number of radial points to be a multiple of 2^{l-1} where l is the number of levels. To ensure this condition is satisfied, after doubling, the density in the region $[0.8, 1.0]$ is increased until this condition is satisfied. Finally, in the third case denoted by "2 refinements", the number of points in the region of interest is quadrupled, with similar adjustments made in the GmgPolar case.

Figure 6.15 shows the convergence of the L_∞ norm of the error for this case. We see that all the solvers benefit from the refinement. The Spline FEM solver introduces the

smallest number of additional points for the most gain. The quadratic splines show more impressive results than the cubic splines as the higher order of the cubic splines means that they already provide an accurate representation of the solution with a single refinement. Fewer points are introduced here as compared to the GmgPolar solver as the Spline FEM solver does not have to respect any additional restrictions regarding the number of points. The Embedded Boundary solver introduces the most points per level as the patches refine in two dimensions instead of one. Additionally, in order to refine radially, square patches are chosen which overlap with the desired domain $r \in [0.8, 1.0]$. The refined domain is therefore larger than necessary. As a result, the large gains obtained for the Embedded Boundary solver through the refinement do not seem significant, especially in the case of "2 refinements". Were this method to be used in a plasma simulation, a study of the choice of the refinement domain should be conducted to obtain these benefits without increasing the number of points as dramatically.

Figure 6.16 shows the convergence of the L_2 norm of the error for the same case. The results are similar to the results described for the L_∞ norm in Figure 6.15. There is a notable difference in the case of the Embedded Boundary solver. When examining the L_2 norm in the "2 refinements" case, we see that the increase in the number of points has very little effect on the L_2 norm despite the fact that the L_∞ norm decreased. This is because the area with large errors is quite small compared to the rest of the simulation. Once the gains have been achieved through the first refinement, further refinement has negligible effect on the L_2 norm as the refined area is no longer a major contributor to the total error.

Finally in Figure 6.17, the errors for the different solvers are compared. The "2 refinements" case is used for the Spline FEM solver and the GmgPolar solver, however as the Embedded Boundary solver does not seem to benefit from a second refinement, the "1 refinement" case is used here. We see that the Embedded Boundary solver is no longer the least accurate solver. This is due to the advantages that come with the ability to refine in two dimensions on patches. While the GmgPolar and the Spline FEM solver can refine poloidally, neither solver can do this exclusively in the outer region where the smaller scale structures are found.

6.5.5. Code Usability

In addition to the measurable differences which are compared above, there are additional points which should be taken into consideration when choosing which solver is best adapted to the GYSELA code. One important consideration is the potential difficulties which may be encountered when trying to couple the solver to the GYSELA code. All the solvers use existing libraries. The Embedded Boundary solver is based on the AMReX library(Zhang, A. Almgren, Beckner, et al. 2019), the Spline FEM solver is based on the SeLaLib library(SeLaLib Development Team 2018), and the GmgPolar solver uses the MUMPS library(Amestoy, Duff, L'Excellent, et al. 2001). On some supercomputers, it can be difficult to access all the necessary tools to couple large complicated libraries to existing codes. This is especially bothersome in the case of

codes such as GYSELA which are regularly run on a variety of different machines. In the case of the SeLaLib library this problem is minimal for two reasons. First, it is not the whole library which must be coupled, but just the relevant modules. Secondly, the SeLaLib library was designed to provide modules for the GYSELA code, so a minimum of intercompatibility is to be expected. In contrast, the AMReX library is quite large and would need to be fully integrated into the GYSELA code in order to compile and use the Embedded Boundary solver. This may make it complicated to couple the two. The MUMPS library would also need coupling, but this library is a common tool which is pre-installed on most supercomputers.

Another important consideration is the choice of grid points. Only the GmgPolar solver allows the user to choose exactly where they would like the grid points to be located. The Embedded Boundary solver requires that the points be equidistant on each level, although additional refinement can be added on uniform patches. The Spline FEM solver allows the user to choose the position of the knots of the spline, but not the position of the points themselves. In special cases (odd degree, uniform knots), the majority of the interpolation points coincide with the knots, however in the general case the user defines the knots and the points are deduced as Greville abcissae. This does however allow the user to specify where there should be refinement. As the GYSELA code evolves on logical coordinates, this means that only the GmgPolar code could be coupled to it without requiring additional steps to move between the points used by GYSELA and the points used by the solver. As the Spline FEM solver also evolves on a logical grid, the problem could be avoided in this case if GYSELA decides to define its points as Greville abcissae.

An additional consideration in the choice of grid points is their number. This time it is the Spline FEM solver which is the least restrictive. As both the Embedded Boundary solver and the GmgPolar solver use multigrid methods, the number of points in a given direction n must be chosen to ensure the number of levels l is sufficiently large. We can determine the maximum number of levels possible for a given problem by writing the number of points as

$$n = N_C \cdot C2^{l-1}, \tag{6.32}$$

where N_C is the minimum number of points in each direction on the coarsest grid, generally $N_C = 2$, C is a constant not divisible by two. Indeed if C is too large, the conjugate gradient method, used to solve the coarsest level in the Embedded Boundary solver, fails to converge. This must be taken into consideration when choosing the grid points, and can be an especially cumbersome restriction when trying to add local refinement. The Embedded Boundary solver avoids this difficulty by defining refinements on patches which are on separate multigrid levels, however for the GmgPolar solver refinement in a given area is often more complicated than simply doubling the number of grid points in this area. In contrast, the Spline FEM solver has no restrictions on the number of knots that can be supplied.

6.6. Culham Geometry

We will now apply our three solvers to the non-analytical "Culham geometry" (Connor, Cowley, Hastie, et al. 1988). As mentioned in Section 6.1,this geometry provides a good description of the cross-section of a tokamak and has been chosen to take into account more realistic geometry in GYSELA. The geometry of a plasma is defined through the shape of the magnetic flux surfaces of an MHD equilibrium satisfying the Grad-Shafranov equation. This geometry is a valid solution to this equation in the limit $\varepsilon = \frac{a}{R_0} \to 0$, where a and R_0 are respectively the minor and major radius of a tokamak. In this section, we will describe this geometry and show that the solvers all function correctly on it.

The mapping describes the electromagnetic equilibrium in a tokamak. It is calculated using a Taylor expansion on the small parameter $\varepsilon \ll 1$. As previously, we will consider normalised coordinates such that $a = 1$. The terms in the mapping are accurate to $\mathcal{O}(\varepsilon^2)$. The mapping is defined as follows:

$$\begin{aligned} x(r,\theta) &= r\cos(\theta) - E(r)\cos(\theta) + T(r)\cos(2\theta) - P(r)\cos(\theta) + \delta(r) + R_0 \\ y(r,\theta) &= r\sin(\theta) + E(r)\sin(\theta) - T(r)\sin(2\theta) - P(r)\sin(\theta) \end{aligned} \tag{6.33}$$

where $E(r)$, $T(r)$, and $\delta(r)$ are functions controlling respectively the elongation, triangularity, and Shafranov shift. These terms are considered to be of order $R_0\varepsilon^2$. $P(r)$ is an additional term of order $R_0\varepsilon^3$ used to ensure that the transformation is quasi-toroidal, and is defined as follows:

$$P(r) = \frac{r^3}{8R_0^2} + \frac{r\delta(r)}{2R_0} - \frac{E(r)^2}{2r} - \frac{T(r)^2}{r}. \tag{6.34}$$

The geometric properties are defined as follows:

$$E''(r) + \left(\frac{1}{r} + \frac{2f'(r)}{f(r)}\right)E'(r) - 3\frac{E(r)}{r^2} = 0, \tag{6.35}$$

$$T''(r) + \left(\frac{1}{r} + \frac{2f'(r)}{f(r)}\right)T'(r) - 8\frac{T(r)}{r^2} = 0, \tag{6.36}$$

$$\delta'(r) = -\frac{1}{R_0 r^2 f(r)^2}\left(\int_0^r r' f(r')^2 dr' - \int_0^r \frac{2r'^2\mu_0 p'(r')}{B_0^2} dr'\right), \tag{6.37}$$

where μ_0 is the magnetic constant, $p(r)$ is the plasma pressure, and $f(r)$ and B_0 are terms used to define the magnetic field $B(r)$:

$$B(r) = B_0 R_0 \left(f(r)\hat{e}_\theta + g(r)\hat{e}_\phi\right), \tag{6.38}$$

where $\hat{e}_\theta$ and $\hat{e}_\phi$ are the unit vectors in the poloidal and toroidal directions. The integration constants of the functions $E(r)$, $T(r)$ and $\delta(r)$ are defined using the constants C_E and C_T such that $E(a) = C_E$, $T(a) = C_T$, and $\delta(a) = 0$.

The functions $f(r)$ and $g(r)$ are approximated from the following system of equa-

tions:

$$f(r) = \zeta(r) g(r), \tag{6.39}$$

$$\zeta(r) = \frac{r}{q(r) R_0}, \tag{6.40}$$

$$g'(r) = \frac{\left(\frac{\zeta(r)}{r} + \zeta'(r)\right) g(r) + \frac{\mu_0 p'(r)}{B_0^2 p_0 f(r)}}{\zeta(r) + \frac{1}{f(r)}}, \tag{6.41}$$

where $q(r)$ is the classical safety factor in the large aspect ratio approximated by the following equation:

$$q(r) = q_0 + (q_a - q_0) r^2, \tag{6.42}$$

where $q_0 = q(0)$ and $q_a = q(a)$. In our investigations, we will use the following definition of the plasma pressure:

$$p(r) = p_a + (p_0 - p_a)\left(1 - r^2\right)^\gamma, \tag{6.43}$$

where γ is a constant, and p_0 and p_a are the pressures at respectively $r = 0$ and $r = a$. Equations (6.35), (6.36), and (6.41) are solved using a fourth order Runge-Kutta method with 1000 equidistant radial points and the following initial conditions:

$$E(r_0) = r_0, \tag{6.44}$$

$$E'(r_0) = 1, \tag{6.45}$$

$$T(r_0) = r_0^2, \tag{6.46}$$

$$T'(r_0) = 2r_0, g(r_0) = 1. \tag{6.47}$$

The integrals required for the definition of the Shafranov shift are calculated using the trapezoidal rule. Values not on the final grid of calculated values are calculated using linear interpolations.

In order to test the three solvers, we will use the values in Table 6.2 as the parameters defining the "Culham geometry".

E_a	0.25	T_a	0.1	q_0	0.8	q_a	0.7	γ	1.0
p_0	10^5	p_a	10^4	B_0	1.0	R_0	5.0		

Table 6.2.: Parameters used to define the "Culham geometry" in Equations (6.33) - (6.43)

Figure 6.18 shows the result of solving Equation (6.1) with $\alpha(r)$ defined by Equation (6.23), $\beta(r) = 1/\alpha(r)$, and the right hand side $f(x, y)$ defined in the same way as the solution in the multi-scale example defined in Equation (6.29). The experiment is run with a number of points similar to a typical GYSELA simulation, i.e. 512 radial points and 1 024 poloidal points. In the case of the Embedded Boundary solver which does not use curvilinear coordinates and therefore does not have a dimension requir-

ing more points, 1 024 points are used in both the x-direction and the y-direction. As the Embedded Boundary solver has larger errors than the other solvers, in a production code it is likely that we would take a cautious approach by using more, rather than fewer points. In addition, the Embedded Boundary solver is fast with reasonable memory consumption so using more points is not excessively restrictive.

Code	Number of Points	Max Memory (MB)	Time (s)
Spline FEM solver			
degree 2	512×1024	792.0	53.23 ± 0.64
degree 3		1357	75.36 ± 0.56
GmgPolar solver			
no extrapolation, matrix-free	512×1024	147.5	16.23 ± 0.02
implicit extrapolation, matrix-free		128.6	40.21 ± 0.04
implicit extrapolation, with matrix		268.8	1.41 ± 0.09
Embedded Boundary solver	1024×1024	396.6	1.53 ± 0.01

Table 6.3.: Comparison of the performance of the three solvers on the "Culham geometry" with the right hand side defined by Equation (6.29).

Table 6.3 shows the results from using the three solvers. As in Section 6.5, we observe that the GmgPolar solver has the lowest memory requirements, and the Embedded Boundary solver is the fastest. The GmgPolar solver could be sped up to a similar execution time as the Embedded Boundary solver by using assembled matrices, at the price of a greater memory consumption. The error cannot be evaluated for this case as the exact solution is unknown. All solvers produce comparable results, with the same L_∞ norm of the solution: $1.30 \cdot 10^{-3}$.

6.7. X-Point Geometry

Ideally, tokamak simulations would like to model not just the core, but also the edge of the plasma. This introduces two additional problems which have not been considered in the rest of this chapter. Firstly, the boundary can have a more complicated shape. It will not follow the geometry of the system, which is chosen to describe the closed magnetic field lines. The Spline FEM solver and the GmgPolar solver in their current form cannot describe a boundary with a more complex geometry without additional developments. However, the embedded boundary used by the Embedded Boundary solver is specifically designed to handle this problem.

For simulations reaching the edge of the plasma, the magnetic field lines are not all closed. At the transition between open and closed field lines, a second singular point appears. This is the source of the second problem. However, this problem

only becomes critical if the singular point appears explicitly in the mapping. If the GmgPolar solver or the Spline FEM solver were modified to include a method for handling the boundary, then they would extend the chosen curvilinear coordinate system outwards. As these coordinates do not have a singular point at the so called x-point, the singularity should not be problematic. However, the choice of these curvilinear coordinates is somewhat arbitrary outside the LCFS where they no longer describe the magnetic field lines.

Figure 6.19 shows the results obtained using the Embedded Boundary solver with a right hand side defined by Equation (6.29). In the central region, the geometry is approximated by the "Culham geometry". In the outer region, including the divertor region, a constant extrapolation of $\alpha(r)$, $\beta(r)$ and $f(x,y)$ is used to define the values. The boundary is defined using the analytical solution to the Grad–Shafranov equation proposed by Cerfon and Freidberg 2010. The configuration representing a lower single null National Spherical Torus Experiment (NSTX)-like equilibrium is chosen. The final boundary is obtained by adding a buffer of size 0.15 around the equilibrium. The equation describing the equilibrium is detailed in D.

As we can see the boundary is not convex. This presents difficulties for the embedded boundary scheme. In each cell, it is assumed that the boundary can be approximated by a straight line. This assumption breaks down at the inflection points near the X-point. To generate the results shown in Figure 6.19, the grid was carefully chosen such that the inflection points are found on the grid. This avoids the problem but is not ideal as the points must be known to a high precision. Additionally, as three points must be found on the grid it puts large constraints on the choice of grid. As mentioned previously, the total number of points n in a direction is written as $n = C2^{l-1}$ where C is as small as possible. Therefore, the position of the inflection points dictates the boundaries of the simulation. Furthermore, as the points must be on grid points at each viable level, the total number of levels is also restricted.

In order to use a X-point boundary in a plasma simulation, methods capable of handling convex boundaries and their effects on the convergence should be investigated. Other codes handling X-point geometry have also encountered these problems. In Jorek (Hoelzl, G.T.A. Huijsmans, Pamela, et al. 2021), flux-surface aligned grids are used to avoid this problem. The domain is split into multiple patches to avoid handling open and closed field-lines simultaneously. In Soledge3X (Bufferand, Balbin, Baschetti, et al. 2022) the boundary is represented by a step function following the edges of the cells. This could be implemented using the Embedded Boundary solver, however this method may have an effect on the convergence, especially in the case of a multigrid method which will struggle to describe the geometry effectively with a 0-th order method on the least refined levels. Further investigations should therefore be conducted to determine the method the best adapted to the methods presented in this chapter.

Conclusion

We have presented three solvers for the gyrokinetic Poisson-like equation defined by Equation (6.1) on geometries of increasing complexity, which represent a polar cross-section of a tokamak reactor. The two most common choices of coordinates in tokamak simulations were considered, namely Cartesian coordinates, and flux-aligned curvilinear coordinates; the latter introduce an artificial singularity at the magnetic axis which requires special care. The first solver, known as the Spline FEM solver, uses isogeometric analysis, in other words spline finite elements, on a geometry defined through a spline mapping, parameterised with the aforementioned curvilinear coordinates. The singularity at the pole is handled through the use of so-called "polar splines" which ensure C^1 smoothness of the solution. This allows the scheme to have a flexible order of convergence which depends on the degree of the splines. The matrices are solved using a preconditioned conjugate gradient scheme. The second, known as the GmgPolar solver, uses finite differences to solve the equation on curvilinear coordinates. The resulting matrices are solved using a multigrid method with the possibility to use an implicit extrapolation scheme to increase the approximation order of the method. The singular point is handled through the discretisation. Finally, the third solver, known as the Embedded Boundary solver, uses second order finite volumes to solve the equation on Cartesian coordinates. This choice of coordinates does not lead to an artificial singularity but means that the boundaries are no longer found at the grid points. Instead, the physical boundary is defined by an embedded boundary approach. The resulting linear system is then solved with a geometric multigrid method in a matrix-free implementation.

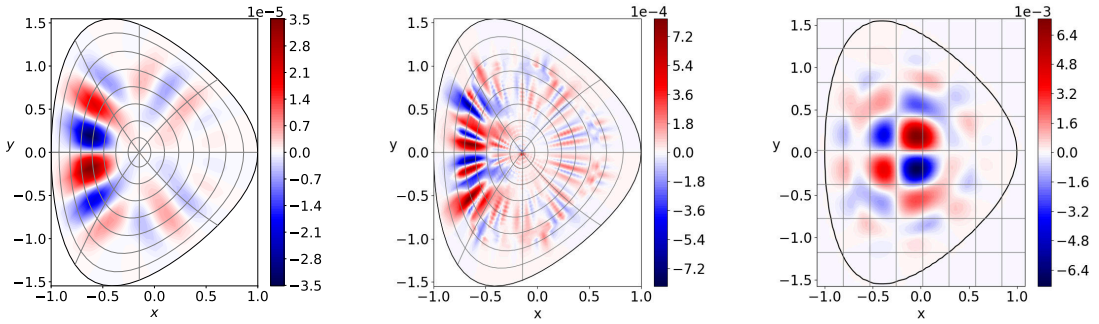
Performance and error criteria were used to compare these solvers on analytical problems and more realistic cases including the so-called "Culham geometry" (Connor, Cowley, Hastie, et al. 1988), and an X-point boundary. The Embedded Boundary solver was found to be the fastest and have the most effective parallelisation (due to its implementation using the AMReX library (Zhang, A. Almgren, Beckner, et al. 2019)). The Spline FEM solver was found to reduce the error the most thanks to its higher order scheme, although this did not necessarily compensate for its heavy memory usage and slow execution time. It is possible that improvements could be made on this aspect, for example by using an efficient sparse solver instead of the preconditioned conjugate gradient method. The GmgPolar solver was found to have the smallest memory requirements thanks to its matrix-free implementation, and is a compromise between relatively fast execution and high order of approximation. If the focus is on execution speed and memory limitations are not crucial, the assembled matrix version of GmgPolar solver may be favourable as it speeds up the serial execution by a factor of seven.

All three solvers allow refinement in troublesome areas. In the GmgPolar solver and the Spline FEM solver this is achieved by allowing non-uniform points in the polar coordinates. In the Embedded Boundary solver this is achieved via patches. The Spline FEM solver was shown to benefit the most from additional refinement, but all solvers demonstrated improved performance when refining in numerically

troublesome areas.

While all solvers were capable of handling the realistic "Culham geometry", at this stage only the Embedded Boundary solver was capable of handling an X-point geometry. A small example was presented in Section 6.7, however this served to highlight a difficulty facing any solver which aims to tackle this geometry in a simulation code; namely the handling of a concave boundary.

Our conclusions so far concern the state-of-the-art of the three solvers and their current implementation. The results suggest that high order and efficient solution of the linear system—both in terms of algorithm and implementation—are two key ingredients for an efficient solver. We note also, that each of the three solvers presented can be improved in at least one of these two directions.

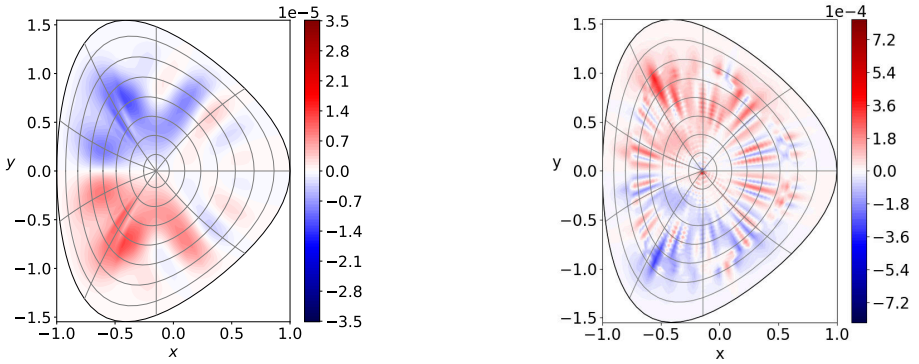


(a) Cubic Spline FEM solver, the L_∞ norm of the error is $3.51 \cdot 10^{-5}$

(b) GmgPolar solver, the L_∞ norm of the error is $8.29 \cdot 10^{-4}$

(c) Embedded Boundary solver, the L_∞ norm of the error is $7.27 \cdot 10^{-3}$

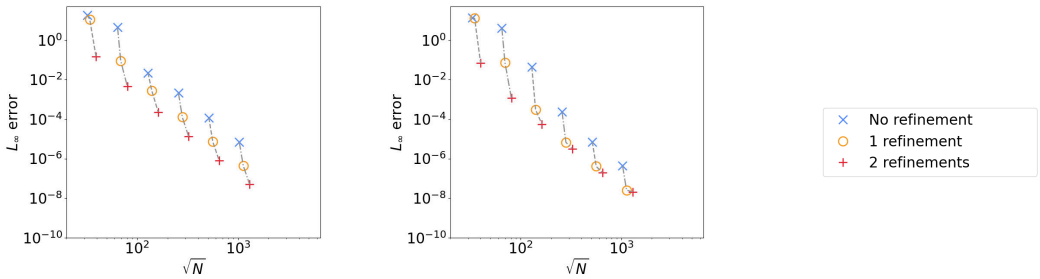
Figure 6.13.: The error, normalised by the ∞-norm of the solution, obtained when solving for Equation (6.28) on the Czarny geometry defined by (6.26) with 64×64 points.



(a) Cubic Spline FEM solver, the L_∞ norm of the error is $1.29 \cdot 10^{-5}$

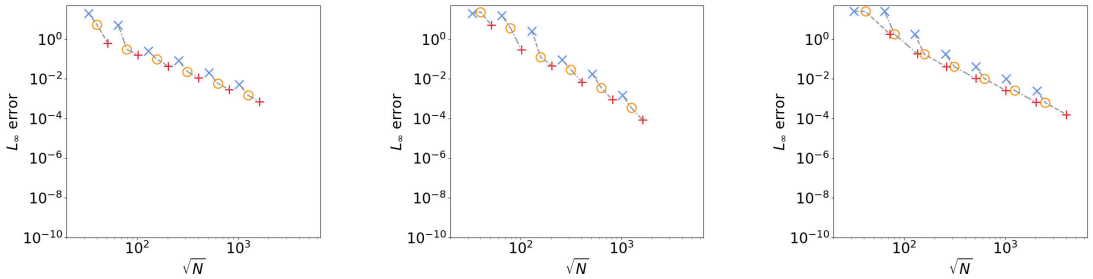
(b) GmgPolar solver, the L_∞ norm of the error is $6.91 \cdot 10^{-4}$

Figure 6.14.: The error, normalised by the ∞-norm of the solution, obtained when solving for Equation (6.28) on the "Czarny" geometry defined by (6.26) with 64 uniform points in the r-direction and 72 non-uniform points in the θ-direction. The additional points are added such that the point density is 50% larger in the region $\theta = [-\pi/4, \pi/4]$



(a) Quadratic Spline FEM solver

(b) Cubic Spline FEM solver

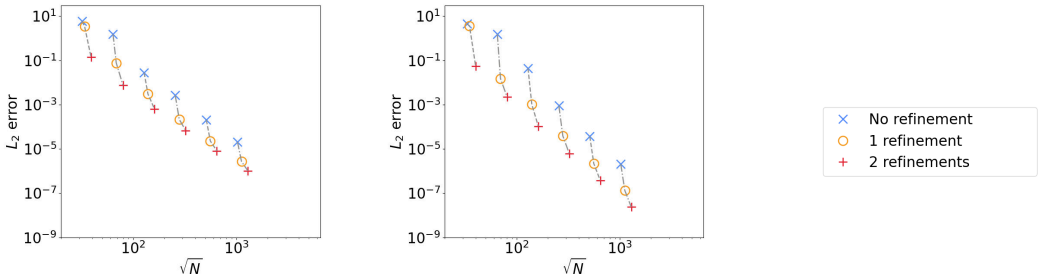


(c) GmgPolar solver without extrapolation

(d) GmgPolar solver with implicit extrapolation

(e) Embedded Boundary solver

Figure 6.15.: L_∞ norm of the error, normalised by the ∞-norm of the solution, obtained when solving for the multi-scale solution described by Equation (6.29) on "Czarny" geometry described by Equation (6.29) with refinement on the domain $r \in [0.8, 1.0]$. Results for the same grid before and after refinement in the region $r \in [0, 0.8]$ are joined by dashed grey lines.



(a) Quadratic Spline FEM solver

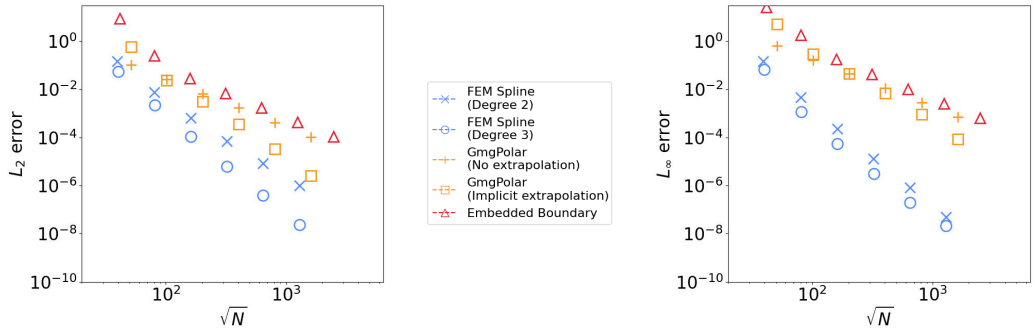
(b) Cubic Spline FEM solver

(c) GmgPolar solver without extrapolation

(d) GmgPolar solver with implicit extrapolation

(e) Embedded Boundary solver

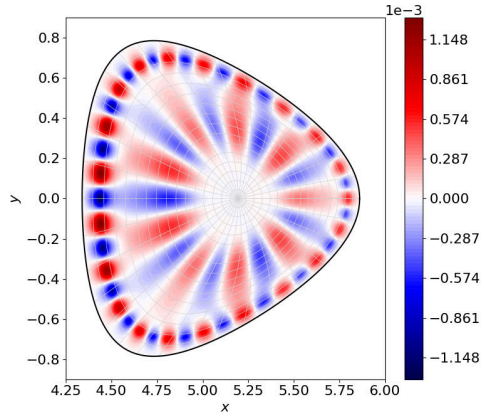
Figure 6.16.: L_2 norm of the error, normalised by the ∞-norm of the solution, obtained when solving for the multi-scale solution described by Equation (6.29) on "Czarny" geometry described by Equation (6.29) with refinement on the domain $r \in [0.8, 1.0]$. Results for the same grid before and after refinement in the region $r \in [0, 0.8]$ are joined by dashed grey lines.



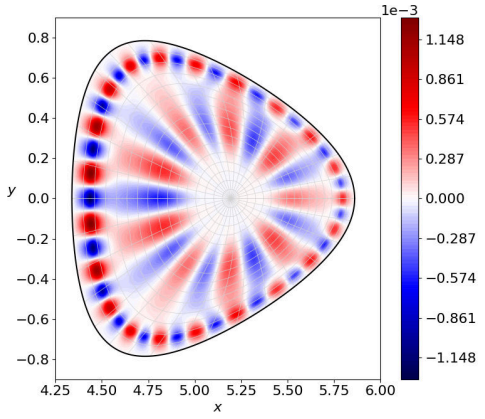
(a) L_2 error

(b) L_∞ error

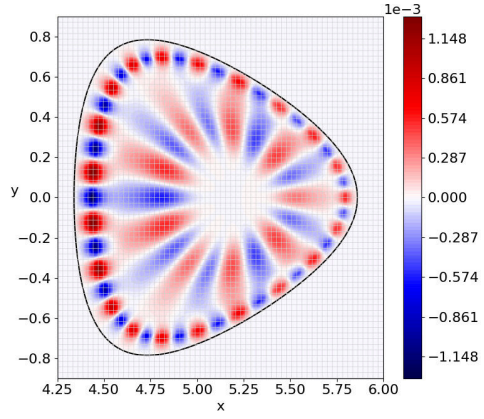
Figure 6.17.: Errors, normalised by the ∞-norm of the solution, when solving for Equation (6.29) on "Czarny" geometry defined by Equation (6.26) with different amounts of refinement in the region $r \in [0.8, 1.0]$. The Spline FEM solver and the GmgPolar solver are four times more refined radially in the region $r \in [0.8, 1.0]$ than in the region $r \in [0, 0.8]$. The Embedded Boundary solver is twice as refined in both directions in the region $r \in [0.8, 1.0]$.



(a) Solution calculated on "Culham geometry" with the Spline FEM solver



(b) Solution calculated on "Culham geometry" with the GmgPolar solver



(c) Solution calculated on "Culham geometry" with the Embedded Boundary solver

Figure 6.18.: The solution to Equation (6.1) with $\alpha(r)$ defined by Equation (6.23), $\beta(r) = 1/\alpha(r)$, and the right hand side $f(x, y)$ defined in the same way as the solution in the multi-scale example defined in Equation (6.29) on the "Culham geometry".

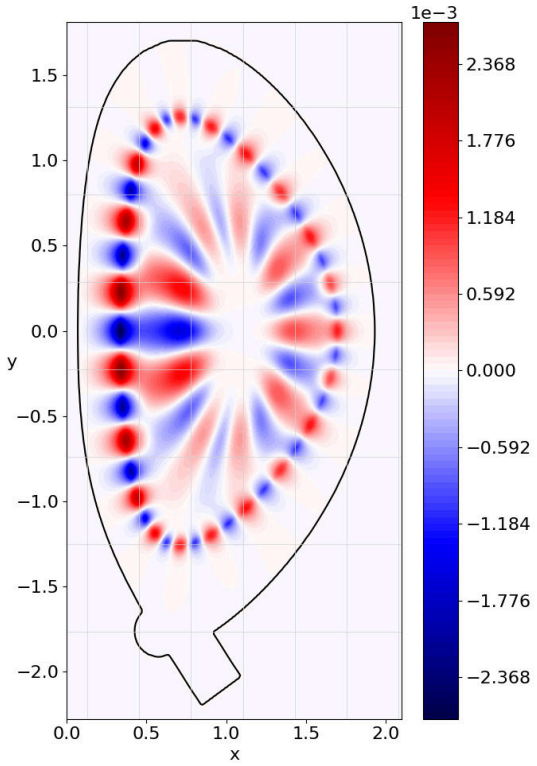


Figure 6.19.: The solution obtained when solving Equation (6.1) with the right hand side defined by Equation (6.29) on an X-Point configuration with 256 points in the x-direction, and 256 points in the y-direction

7. Conclusion

In this thesis I have explained my contributions towards the exascale version of the GYSELA code (V. Grandgirard, Abiteboul, J. Bigot, et al. 2016). This future version of GYSELA includes a treatment of both the core and edge regions of the plasma. In the edge region, two problems arise that have been neglected in the GYSELA code up to now. Firstly there are steeper gradients (Guilhem Dif-Pradalier, Philippe Ghendrih, Yanick Sarazin, et al. 2022), and secondly the geometry can rarely be realistically modelled by a circular geometry. This thesis fits into this context and discusses potential solutions to these problems.

When simulations contain steep gradients it is important to sufficiently refine these features. However the sizes of the problems currently tackled by GYSELA, are already so large that a significant increase to the refinement would lead to a simulation which is too memory-heavy to run on the existing petascale supercomputers. An obvious solution to this problem is to use non-uniform points.

There are at two ways to add non-uniform points to a code: the various schemes can be adapted to allow for non-uniform points, or patches can be used to separate the domain into areas which will be treated differently. The best choice for a code depends on the methods that it uses. The GYSELA code is based on the semi-Lagrangian method, with the distribution function being approximated using splines at each time step. As I explained in Chapter 2, there is nothing in the theory of splines which precludes their use for non-uniform problems. However the use of uniform knots to define these splines allows simplifications to be made to the evaluation method (described in Section 2.6) which result in faster code. On the other hand, a spline problem is inherently global. This makes the use of patches non-trivial. Boundary conditions are introduced at the intersection of different patches which must be handled in some way. This splitting of a spline domain results in local splines whose theory is not yet fully investigated.

In Chapter 3 and (Bourne, Munschy, Virginie Grandgirard, et al. 2022), I investigated the use of non-uniform splines as a way of increasing the number of points in a given region. In order to test this, the context of sheath simulations was chosen. The plasma sheath is the region of a plasma adjacent to a wall. This region exhibits particularly steep gradients which require extremely fine refinement. As a first step this method was not tested in the full GYSELA code, but rather in a mini-application, known as VOICE. VOICE solves the Vlasov-Poisson equations in 1D-1V (one spatial dimension, and one velocity dimension).

In order to adapt the VOICE code to run with non-uniform points, all the schemes used had to be adapted to allow for non-uniform points. As splines are not inherently uniform, it is often simpler to express schemes via these tools instead of deriving a

non-uniform expression. This was seen in Chapter 2 with the example of quadrature. The non-uniform Newton-Cotes schemes detailed in Section 2.8.2.3 are very long which makes them prone to implementation errors. However the spline derivation of a quadrature scheme can be expressed in a more compact manner. Furthermore the "best" quadrature (using Schoenberg's criteria, Schoenberg 1964) is intrinsically linked to a spline problem.

In Chapter 2 I presented a new method for calculating the coefficients of this "best" quadrature scheme. This method improved the conditioning of the problem, providing improved precision compared to a naive approach based on b-spline derivatives. For example, in the case of a fifth order scheme, four extra significant figures of precision are obtained. As a result the calculated coefficients are sufficiently accurate to be used for high precision quadrature even when using a high order scheme. This method, presented in Section 2.8.2.1, represents essentially an improved handling of the boundary conditions, such that the sparsity pattern of the matrix is more favourable. I therefore believe that this result could be extended to help with the conditioning of the interpolation problem for high-order splines with Hermite boundary conditions which also exhibit high condition numbers. This should be particularly easy in the case of uniform splines where the normalisation coefficient can be calculated trivially. This question could be explored in further work.

Having equipped VOICE with the necessary non-uniform schemes, tests were then run to investigate the constraints associated with non-uniform points. A judicious choice of non-uniform points were shown to reduce the memory constraints of the sheath simulation by 89%. However a significant numerical cost was incurred due to the use of non-uniform points. On 8 OpenMP threads, non-uniform schemes were shown to be 85% slower than uniform schemes. While this cost was offset by the reduction in points for this case, this may not be the case for problems with gentler gradients. Use of GPU parallelism was, however, shown to help with this problem. Non-uniform schemes were only 30% slower than uniform schemes when GPUs were used to accelerate these methods. In the case of the simulations studied, non-uniform simulations were therefore able to run 5.5 times faster than uniform simulations providing equivalent results.

Chapter 3 also investigated the effect of the chosen degree on the solution. Higher-order schemes were shown to converge quickly, however they also introduced spurious oscillations. In the case of penalised sheath simulations which also contain a numerical representation of the wall, these oscillations easily lead to negative values in the distribution function. This non-physical behaviour should be avoided in areas where the results are analysed. In the case of sheath simulations, it is precisely at the intersection between the wall and the plasma where both the oscillations and the sheath occur. As a result low order schemes are to be preferred in this case.

A low enough degree, and a large enough number of points therefore allowed the avoidance of non-physical behaviour near the boundary. Furthermore these simulations were shown to have good conservation properties. The mass, momentum and energy were all conserved to a high degree of accuracy. These properties, as well as the increased domain size made available through the use of non-uniform points, allowed

Munschy, Bourne, Guilhem Dif-Pradalier, et al. 2022 to investigate the physics of the sheath by running previously unreachable simulations.

The second method for adding non-uniform points to a simulation is the use of patches and local splines. One possibility for the implementation of local splines has already been investigated by Nicolas Crouseilles, Guillaume Latu, and Sonnendrücker 2009, however this method is limited to uniform splines with the same refinement on each patch, rendering it useless for mesh refinement purposes. In Chapter 4, I investigated a new local spline method. Semi-Lagrangian advection on splines constructed via this method was proven to be stable. A parallelisation scheme was also proposed for a Vlasov-Poisson system where both the Vlasov equation and the Poisson equation are distributed. While this scheme seems very promising it cannot be implemented in GYSELA without further tests. The next step for this work will be to implement it in a test code and examine its performance. As described, this method is 1D, however the framework lends itself readily to an N-D implementation. A further step will therefore be to implement a 2D version of these local splines.

The sheath simulation could also be used as a test bed for these further developments of the local splines. Changes would need to be made throughout the VOICE code in order to implement the advection of the boundary derivatives, necessary for the local spline implementation. Rather than modifying large swathes of the code, this opportunity should be used to test the new version of VOICE written in C++. I have already implemented the global splines in this code, the next step is therefore to use these global splines locally to test the described scheme.

The second problem arising in the edge region is the shape of the geometry. In Chapter 5 the changes made to the GYSELA code in order to handle a realistic D-shaped geometry were presented. This new geometry is based on an analytical magnetic equilibrium, known as the "Culham" equilibrium. The repercussions on the code concern the definition of the terms in generalised coordinates and the choice of solver for the Poisson equation. A spline FEM solver was implemented in the GYSELA code and results of GAM tests (Rosenbluth and Hinton 1998) were used to validate the geometry in the GYSELA code. This was done by comparing the results to those obtained by two other gyrokinetic codes: ORB5 (S. Jolliet, Bottino, Angelino, et al. 2007) and GENE (Jenko, Dorland, Kotschenreuther, et al. 2000), using the results obtained by Biancalani, Bottino, Ehrlacher, et al. 2017. The results were also compared to analytic results (Sugama and Watanabe 2006; Gao 2011; Xiao and Catto 2006) however the assumptions used to obtain these results were shown to limit their usefulness.

The solver chosen for the new geometry needs to be able to handle the non-orthogonality of the chosen geometry, however ideally it also needs to be fast and accurate with low memory requirements. In order to prepare for the future implementation of non-uniform points in the code it also needs to be able to handle such points and be able to benefit from them. In Chapter 6 three possible solvers are compared. They are a spline-based finite elements solver operating on polar coordinates, referred to as the Spline FEM solver (Section 6.2), a geometric multigrid solver operating on polar coordinates, referred to as the GmgPolar solver (Section 6.3), and a finite

differences solver operating on Cartesian coordinates with an embedded boundary approach, referred to as the Embedded Boundary solver (Section 6.4).

All three solvers are shown to be capable of solving the equation on a realistic non-analytical geometry. The Spline FEM solver and the GmgPolar solver, which both use polar coordinates, also implement methods to handle the problem of the singular point. The Spline FEM solver is shown to be the most accurate. The GmgPolar solver is shown to use the least memory. The Embedded Boundary solver is shown to be the fastest in most cases. On balance it seems that the GmgPolar solver provides the best compromise solution for the GYSELA code. Memory is a critical problem, and the GmgPolar solver is the second fastest. Furthermore, through the use of the implicit extrapolation method, its convergence order approaches that of the cubic Spline FEM solver.

All three solvers allow refinement in troublesome areas. In the GmgPolar solver and the Spline FEM solver this is achieved by allowing non-uniform points in the polar coordinates. In the Embedded Boundary solver this is achieved via patches. The Spline FEM solver was shown to benefit significantly more from additional refinement in numerically troublesome areas than the other two solvers, but all solvers demonstrated improved performance. The GmgPolar solver and the Spline FEM solver were however limited by the refinement in poloidal coordinates. As soon as the error near the edge was limited poloidally, additional refinement was unable to improve the error. It would be interesting to see if an efficient local spline version of the Spline FEM solver could be developed based on the work in Chapter 4. This would allow this solver to refine poloidally on a sub-domain.

The Embedded Boundary solver was additionally used to attempt to solve an X-point geometry. If GYSELA wants to one day model the ITER tokamak in its entirety, it will require a method of describing such a geometry. The presence of the singular X-point makes this a particularly difficult challenge. No matter the solution chosen to handle the equations, embedded boundaries will be required to accurately describe the shape of the machine. This test therefore provides an idea of the problems that GYSELA may face in the future if embedded boundaries are used with this, or another, solver. In particular, the concave shape of the boundary poses problems for higher order schemes.

The GmgPolar solver also provides potential for future developments in GYSELA. Although the solver presented is 2D, the theory can easily be extended to a 3D problem. Currently the Poisson solver only solves the problem in two dimensions. This is possible as the poloidal component of the magnetic field lines is neglected when defining the perpendicular gradient as described in Section 5.2.2. The neglected term is second order in an axisymmetric configuration, however should GYSELA ever wish to model a stellarator configuration this term will no longer be negligible. It is therefore useful to have a solution to this problem to hand.

This thesis is therefore a mixture of theoretical work (including the proof of stability of the semi-Lagrangian method on local splines in Chapter 4, and the expression of equations in generalised coordinates in Section 5.2.1), numerical work (including the study of non-uniform splines in Chapter 3 and solvers of a Poisson-like equation in

Chapter 6), and computer science (including the implementations of the different methods in Fortran, C++ and python, and the implementation of GPU routines); all this in close collaboration with plasma physicists.

Bibliography

[Afe+14] Bedros Afeyan, Fernando Casas, Nicolas Crouseilles, et al. “Simulations of kinetic electrostatic electron nonlinear (KEEN) waves with variable velocity resolution grids and high-order time-splitting”. In: *The European Physical Journal D* 68.10 (Oct. 2014), p. 295. ISSN: 1434-6079. DOI: 10.1140/epjd/e2014-50212-6. URL: https://doi.org/10.1140/epjd/e2014-50212-6 (cit. on pp. 73, 81).

[al19] Zhang et al. “AMReX: A Framework for Block-Structured Adaptive Mesh Refinement”. In: *Journal of Open Source Software* 4.37 (2019), p. 1370. DOI: 10.21105/joss.01370 (cit. on pp. 8, 10, 33, 156).

[Alm+98] Ann S. Almgren, John B. Bell, Phillip Colella, et al. “A Conservative Adaptive Projection Method for the Variable Density Incompressible Navier–Stokes Equations”. In: *Journal of Computational Physics* 142.1 (1998), pp. 1–46. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1998.5890 (cit. on p. 157).

[Ame+01] Patrick R Amestoy, Iain S Duff, Jean-Yves L’Excellent, et al. “A fully asynchronous multifrontal solver using distributed dynamic scheduling”. In: *SIAM Journal on Matrix Analysis and Applications* 23.1 (2001), pp. 15–41 (cit. on pp. 155, 170).

[AMR+22] the AMReX Development Team, Ann Almgren, Vince Beckner, et al. *AMReX-Codes/amrex: AMReX 22.06*. Version 22.06. June 2022. DOI: 10.5281/zenodo.6603626. URL: https://doi.org/10.5281/zenodo.6603626 (cit. on p. 156).

[And+99] E. Anderson, Z. Bai, C. Bischof, et al. *LAPACK Users’ Guide*. Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999. ISBN: 0-89871-447-8 (paperback) (cit. on p. 151).

[Ang+09] P. Angelino, X. Garbet, L. Villard, et al. “Role of Plasma Elongation on Turbulent Transport in Magnetically Confined Plasmas”. In: *Phys. Rev. Lett.* 102 (19 May 2009), p. 195002. DOI: 10.1103/PhysRevLett.102.195002. URL: https://link.aps.org/doi/10.1103/PhysRevLett.102.195002 (cit. on p. 122).

[BMN18] Badsi, M., Mehrenberger, M., and Navoret, L. “Numerical stability of a plasma sheath”. In: *ESAIM: ProcS* 64 (2018), pp. 17–36. DOI: 10.1051/proc/201864017. URL: https://doi.org/10.1051/proc/201864017 (cit. on p. 72).

[BJL11] David Bailey, Karthik Jeyabalan, and Sherry Li. "A Comparison of Three High-Precision Quadrature Schemes". In: *Experimental Mathematics* 14 (Jan. 2011). DOI: 10.1080/10586458.2005.10128931 (cit. on p. 68).

[Bar88] Saulo RM Barros. "The Poisson equation on the unit disk: a multigrid solver using polar coordinates". In: *Applied Mathematics and Computation* 25.2 (1988), pp. 123–135 (cit. on pp. 152, 154).

[BM82] J. R. Bates and A. McDonald. "Multiply-Upstream, Semi-Lagrangian Advective Schemes: Analysis and Application to a Multi-Level Primitive Equation Model". In: *Monthly Weather Review* 110.12 (1982), pp. 1831–1842. DOI: 10.1175/1520-0493(1982)110<1831:MUSLAS>2.0.CO;2. URL: https://journals.ametsoc.org/view/journals/mwre/110/12/1520-0493_1982_110_1831_muslas_2_0_co_2.xml (cit. on p. 101).

[BH12] Marsha Berger and Christiane Helzel. "A Simplified h-box Method for Embedded Boundary Grids". In: *SIAM Journal on Scientific Computing* 34.2 (2012), A861–A888. DOI: 10.1137/110829398 (cit. on p. 156).

[Bes04] Nicolas Besse. "Convergence of a semi-Lagrangian scheme for the one-dimensional Vlasov–Poisson system". In: *SIAM Journal on Numerical Analysis* 42.1 (2004), pp. 350–382 (cit. on p. 101).

[Bes08] Nicolas Besse. "Convergence of a High-Order Semi-Lagrangian Scheme with Propagation of Gradients for the One-Dimensional Vlasov–Poisson System". In: *SIAM Journal on Numerical Analysis* 46.2 (2008), pp. 639–670. DOI: 10.1137/050635171. eprint: https://doi.org/10.1137/050635171. URL: https://doi.org/10.1137/050635171 (cit. on p. 101).

[BM08] Nicolas Besse and Michel Mehrenberger. "Convergence of classes of high-order semi-Lagrangian schemes for the Vlasov–Poisson system". In: *Mathematics of computation* 77.261 (2008), pp. 93–123 (cit. on p. 101).

[BGK54] P. L. Bhatnagar, E. P. Gross, and M. Krook. "A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems". In: *Phys. Rev.* 94.3 (May 1954), pp. 511–525. DOI: 10.1103/PhysRev.94.511. URL: http://link.aps.org/doi/10.1103/PhysRev.94.511 (cit. on p. 76).

[Bia+17] A. Biancalani, A. Bottino, C. Ehrlacher, et al. "Cross-code gyrokinetic verification and benchmark on the linear collisionless dynamics of the geodesic acoustic mode". In: *Physics of Plasmas* 24.6 (2017), p. 062512. DOI: 10.1063/1.4985571. eprint: https://doi.org/10.1063/1.4985571. URL: https://doi.org/10.1063/1.4985571 (cit. on pp. 139, 140, 187).

[BP21] Julien Bigot and Thomas Padioleau. *a Discrete Domain Computation library*. 2021. URL: https://ddc.mdls.fr/ (cit. on p. 33).

[BLS76] Carl de Boor, Tom Lyche, and Larry L. Schumaker. "On Calculating with B-Splines II. Integration". In: *Numerische Methoden der Approximationstheorie/Numerical Methods of Approximation Theory: Vortragsauszüge der Tagung über numerische Methoden der Approximationstheorie vom 25. bis 31. Mai 1975 im Mathematischen Forschungsinstitut Oberwolfach (Schwarzwald)*. Basel: Birkhäuser Basel, 1976, pp. 123–146. ISBN: 978-3-0348-7692-6. DOI: 10.1007/978-3-0348-7692-6_6. URL: https://doi.org/10.1007/978-3-0348-7692-6_6 (cit. on pp. 37, 61).

[BC09] Mihai Bostan and Nicolas Crouseilles. "Convergence of a semi-Lagrangian scheme for the reduced Vlasov–Maxwell system for laser–plasma interaction". In: *Numerische Mathematik* 112.2 (2009), pp. 169–195 (cit. on p. 101).

[Bou18] Emily Bourne. "Parallel gyrokinetic simulations with Python". MA thesis. Technische Universität München, 2018. URL: https://github.com/pyccel/pygyro/blob/master/doc/ThesisPaper/Thesis.pdf (cit. on p. 33).

[Bou+22a] Emily Bourne, Yaman Güçlü, Said Hadjout, et al. "Pyccel: a Python-to-X transpiler for scientific high-performance computing". In: *Journal of Open Source Software* (Nov. 2022). under review (cit. on p. 5).

[Bou+22b] Emily Bourne, Yaman Güçlü, Saïd Hadjout, et al. *Pyccel : Python extension language using accelerators*. 2022. URL: https://github.com/pyccel/pyccel/ (cit. on p. 33).

[Bou+22c] Emily Bourne, Philippe Leleux, Katharina Kormann, et al. "Solver comparison for Poisson-like equations on tokamak geometries". In: *Journal of Computational Physics* (Sept. 2022). submitted. URL: https://hal-cea.archives-ouvertes.fr/cea-03786723 (cit. on pp. 5, 13, 33, 145).

[Bou+22d] Emily Bourne, Yann Munschy, Virginie Grandgirard, et al. "Non-Uniform Splines for Semi-Lagrangian Kinetic Simulations of the Plasma Sheath". In: *Journal of Computational Physics* (Aug. 2022). under review (https://track.authorhub.elsevier.com/?uuid=b7ad331b-bd6c-4e85-834d-2c117471df74). URL: https://hal-cea.archives-ouvertes.fr/cea-03748016 (cit. on pp. 5, 32, 71, 113, 185).

[Bou+18] Nicolas Bouzat, Camilla Bressan, Virginie Grandgirard, et al. "Targeting realistic geometry in Tokamak code Gysela". In: *ESAIM: Proceedings and Surveys* 63 (2018), pp. 179–207 (cit. on pp. 145, 160).

[BHM00] William L Briggs, Van Emden Henson, and Steve F McCormick. *A multigrid tutorial*. SIAM, 2000 (cit. on p. 153).

[Buf+22] H Bufferand, J Balbin, S Baschetti, et al. "Implementation of multi-component Zhdanov closure in SOLEDGE3X". In: *Plasma Physics and Controlled Fusion* 64.5 (Mar. 2022), p. 055001. DOI: 10.1088/1361-6587/ac4fac. URL: https://doi.org/10.1088/1361-6587/ac4fac (cit. on p. 175).

[Buf+21] H. Bufferand, J. Bucalossi, G. Ciraolo, et al. “Progress in edge plasma turbulence modelling—hierarchy of models from 2D transport application to 3D fluid simulations in realistic tokamak geometry”. In: *Nuclear Fusion* 61.11 (Oct. 2021), p. 116052. DOI: 10.1088/1741-4326/ac2873. URL: https://doi.org/10.1088/1741-4326/ac2873 (cit. on p. 29).

[Bur+15] E. Burman, S. Claus, P. Hansbo, et al. “CutFEM: Discretizing geometry and partial differential equations”. In: *International Journal for Numerical Methods in Engineering* 104.7 (2015), pp. 472–501. DOI: 10.1002/nme.4823 (cit. on p. 156).

[CM08] Martin Campos Pinto and Michel Mehrenberger. “Convergence of an adaptive semi-Lagrangian scheme for the Vlasov-Poisson system”. In: *Numerische Mathematik* 108.3 (2008), pp. 407–444 (cit. on p. 101).

[Cas+18] E. Caschera, G. Dif-Pradalier, Ph. Ghendrih, et al. “Immersed boundary conditions in global, flux-driven, gyrokinetic simulations”. In: *Journal of Physics: Conference Series* 1125 (Nov. 2018), p. 012006. DOI: 10.1088/1742-6596/1125/1/012006. URL: https://doi.org/10.1088/1742-6596/1125/1/012006 (cit. on p. 77).

[CF10] Antoine J. Cerfon and Jeffrey P. Freidberg. ““One size fits all” analytic solutions to the Grad–Shafranov equation”. In: *Physics of Plasmas* 17.3 (2010), p. 032502. DOI: 10.1063/1.3328818 (cit. on pp. 123, 175).

[CDM13] Frédérique Charles, Bruno Després, and Michel Mehrenberger. “Enhanced convergence estimates for semi-Lagrangian schemes application to the Vlasov–Poisson equation”. In: *SIAM Journal on Numerical Analysis* 51.2 (2013), pp. 840–863 (cit. on p. 101).

[Che16] Francis F. Chen. *Introduction to Plasma Physics and Controlled Fusion*. Springer Cham, 2016. URL: https://doi.org/10.1007/978-3-319-22309-4 (cit. on p. 28).

[CK12] E Ward Cheney and David R Kincaid. *Numerical mathematics and computing*. Cengage Learning, 2012 (cit. on p. 53).

[CK76] Chio-Zong Cheng and Georg Knorr. “The integration of the Vlasov equation in configuration space”. In: *Journal of Computational Physics* 22.3 (1976), pp. 330–351 (cit. on p. 101).

[Con+88] J. W. Connor, S. C. Cowley, R. J. Hastie, et al. “Tearing modes in toroidal geometry”. In: *The Physics of Fluids* 31.3 (1988), pp. 577–590. DOI: 10.1063/1.866840. eprint: https://aip.scitation.org/doi/pdf/10.1063/1.866840. URL: https://aip.scitation.org/doi/abs/10.1063/1.866840 (cit. on pp. 122, 123, 146, 172, 176).

[CM14] David Coulette and Giovanni Manfredi. “An Eulerian Vlasov code for plasma-wall interactions”. In: *Journal of Physics: Conference Series* 561 (Nov. 2014), p. 012005. DOI: 10.1088/1742-6596/561/1/012005. URL: https://doi.org/10.1088/1742-6596/561/1/012005 (cit. on pp. 71, 72).

[CLS09] Nicolas Crouseilles, Guillaume Latu, and Eric Sonnendrücker. “A parallel Vlasov solver based on local cubic spline interpolation on patches”. In: *Journal of Computational Physics* 228.5 (2009), pp. 1429–1446. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2008.10.041. URL: https://www.sciencedirect.com/science/article/pii/S0021999108005652 (cit. on pp. 101, 119, 187).

[CH08] Olivier Czarny and Guido Huysmans. “Bézier surfaces and finite elements for MHD simulations”. In: *Journal of Computational Physics* 227.16 (2008), pp. 7423–7445. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2008.04.001. URL: https://www.sciencedirect.com/science/article/pii/S0021999108002118 (cit. on p. 159).

[De 78] Carl De Boor. *A practical guide to splines*. Vol. 27. springer-verlag New York, 1978 (cit. on p. 35).

[de 72] Carl de Boor. “On calculating with B-splines”. In: *Journal of Approximation Theory* 6.1 (1972), pp. 50–62. ISSN: 0021-9045. DOI: https://doi.org/10.1016/0021-9045(72)90080-9. URL: https://www.sciencedirect.com/science/article/pii/0021904572900809 (cit. on p. 35).

[Dif+11] G. Dif-Pradalier, P.H. Diamond, V. Grandgirard, et al. “Neoclassical physics in full distribution function gyrokinetics”. In: *Phys. Plasmas* 18 (2011), p. 062309. DOI: 10.1063/1.3592652 (cit. on p. 75).

[Dif+22] Guilhem Dif-Pradalier, Philippe Ghendrih, Yanick Sarazin, et al. “Transport barrier onset and edge turbulence shortfall in fusion plasmas”. In: *Communications Physics* 5 (Sept. 2022). DOI: 10.1038/s42005-022-01004-z (cit. on pp. 32, 71, 74, 160, 169, 185).

[EO14] Lukas Einkemmer and Alexander Ostermann. “Convergence analysis of a discontinuous Galerkin/Strang splitting approximation for the Vlasov–Poisson equations”. In: *SIAM Journal on Numerical Analysis* 52.2 (2014), pp. 757–778 (cit. on p. 101).

[Far93] Gerald Farin. *Curves and Surfaces for Computer-Aided Geometric Design (Third Edition)*. Third Edition. Boston: Academic Press, 1993, pp. 157–187. ISBN: 978-0-12-249052-1. DOI: https://doi.org/10.1016/B978-0-12-249052-1.50015-5. URL: https://www.sciencedirect.com/science/article/pii/B9780122490521500155 (cit. on pp. 35, 148).

[FM03] M. Farrashkhalvat and J.P. Miles. “Variational methods and adaptive grid generation”. In: Dec. 2003, pp. 152–179. ISBN: 9780750650588. DOI: 10.1016/B978-075065058-8/50006-X (cit. on p. 86).

[Fil01] Francis Filbet. “Convergence of a finite volume scheme for the Vlasov–Poisson system”. In: *SIAM Journal on Numerical Analysis* 39.4 (2001), pp. 1146–1169 (cit. on p. 101).

[Gao11] Zhe Gao. “Analytical Theory of the Geodesic Acoustic Mode in the Small and Large Orbit Drift Width Limits and its Application in a Study of Plasma Shaping Effect”. In: *Plasma Science and Technology* 13.1 (Feb. 2011), pp. 15–20. DOI: 10.1088/1009-0630/13/1/04. URL: https://doi.org/10.1088/1009-0630/13/1/04 (cit. on pp. 139, 141, 187).

[Gar+10] Xavier Garbet, Yasuhiro Idomura, Laurent Villard, et al. “TOPICAL REVIEW: Gyrokinetic simulations of turbulent transport”. In: *Nuclear Fusion - NUCL FUSION* 50 (Apr. 2010). DOI: 10.1088/0029-5515/50/4/043002 (cit. on p. 30).

[Gar+21] Xavier Garbet, Olivier Panico, R. Varennes, et al. “Zonal instability and wave trapping”. In: *Journal of Physics: Conference Series* 1785 (Feb. 2021), p. 012002. DOI: 10.1088/1742-6596/1785/1/012002 (cit. on p. 5).

[Gib22] Elizabeth Gibney. “Nuclear-fusion reactor smashes energy record”. In: *Nature* 602 (2022). URL: https://www.nature.com/articles/d41586-022-00391-1 (cit. on pp. 28, 29).

[GHL06] John Goodrich, Thomas Hagstrom, and Jens Lorenz. “Hermite methods for hyperbolic initial-boundary value problems”. In: *Mathematics of Computation* 75 (2006). DOI: https://doi.org/10.1090/S0025-5718-05-01808-9. URL: https://www.ams.org/journals/mcom/2006-75-254/S0025-5718-05-01808-9/ (cit. on p. 109).

[GR74] William J. Gordon and Richard F. Riesenfeld. “B-SPLINE CURVES AND SURFACES”. In: *Computer Aided Geometric Design*. Ed. by ROBERT E. BARNHILL and RICHARD F. RIESENFELD. Academic Press, 1974, pp. 95–126. ISBN: 978-0-12-079050-0. DOI: https://doi.org/10.1016/B978-0-12-079050-0.50011-4. URL: http://www.sciencedirect.com/science/article/pii/B9780120790500500114 (cit. on pp. 35, 38).

[Gra+16] V. Grandgirard, J. Abiteboul, J. Bigot, et al. “A 5D gyrokinetic full-f global semi-Lagrangian code for flux-driven ion turbulence simulations”. In: *Computer Physics Communications* 207 (2016), pp. 35–68. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2016.05.007. URL: https://www.sciencedirect.com/science/article/pii/S0010465516301230 (cit. on pp. 7, 9, 30–32, 72, 81, 82, 101, 119, 121, 129, 131, 138, 139, 145, 158, 160, 165, 185).

[HH91] Günther Hämmerlin and Karl-Heinz Hoffman. “Splines”. In: *Numerical Mathematics*. New York, NY: Springer New York, 1991, pp. 229–271. ISBN: 978-1-4612-4442-4. DOI: 10.1007/978-1-4612-4442-4_6. URL: https://doi.org/10.1007/978-1-4612-4442-4_6 (cit. on pp. 35, 49, 62).

[Hat+02] Roman Hatzky, Trach Minh Tran, Axel Könies, et al. “Energy conservation in a nonlinear gyrokinetic particle-in-cell code for ion-temperature-gradient-driven modes in θ-pinch geometry”. In: *Physics of Plasmas* 9.3 (2002), pp. 898–912. DOI: 10.1063/1.1449889. eprint: https://doi.org/10.1063/1.1449889. URL: https://doi.org/10.1063/1.1449889 (cit. on pp. 30, 145).

[Her+11] P. Hertout, C. Boulbe, E. Nardon, et al. “The CEDRES++ equilibrium code and its application to ITER, JT-60SA and Tore Supra”. In: *Fusion Engineering and Design* 86.6–8 (2011). Proceedings of the 26th Symposium of Fusion Technology (SOFT-26), pp. 1045–1048. ISSN: 0920-3796. DOI: http://dx.doi.org/10.1016/j.fusengdes.2011.03.092. URL: http://www.sciencedirect.com/science/article/pii/S0920379611003656 (cit. on p. 33).

[Hoe+21] M. Hoelzl, G.T.A. Huijsmans, S.J.P. Pamela, et al. “The JOREK non-linear extended MHD code and applications to large-scale instabilities and their control in magnetically confined fusion plasmas”. In: *Nuclear Fusion* 61.6 (May 2021), p. 065001. DOI: 10.1088/1741-4326/abf99f. URL: https://doi.org/10.1088/1741-4326/abf99f (cit. on p. 175).

[HC07] Guido Huijsmans and Olivier Czarny. “MHD stability in X-point geometry: Simulation of ELMs”. In: *Nuclear Fusion* 47 (June 2007), p. 659. DOI: 10.1088/0029-5515/47/7/016 (cit. on p. 29).

[Jen+00] F. Jenko, W. Dorland, M. Kotschenreuther, et al. “Electron temperature gradient driven turbulence”. In: *Physics of Plasmas* 7.5 (2000), pp. 1904–1910. DOI: 10.1063/1.874014. eprint: https://doi.org/10.1063/1.874014. URL: https://doi.org/10.1063/1.874014 (cit. on pp. 30, 140, 143, 187).

[Jia86] Rong-Qing Jia. “Spline Interpolation at a Bi-Infinite Knot Sequence”. In: *SIAM J. Numer. Anal.* 23.3 (June 1986), pp. 653–662. ISSN: 0036-1429. DOI: 10.1137/0723041. URL: https://doi.org/10.1137/0723041 (cit. on p. 46).

[JC98] Hans Johansen and Phillip Colella. “A Cartesian Grid Embedded Boundary Method for Poisson’s Equation on Irregular Domains”. In: *Journal of Computational Physics* 147.1 (1998), pp. 60–85. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1998.5965. URL: https://www.sciencedirect.com/science/article/pii/S0021999198959654 (cit. on pp. 156, 157).

[Jol+07] S. Jolliet, A. Bottino, P. Angelino, et al. “A global collisionless PIC code in magnetic coordinates”. In: *Computer Physics Communications* 177.5 (2007), pp. 409–425. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2007.04.006. URL: https://www.sciencedirect.com/science/article/pii/S0010465507002251 (cit. on pp. 30, 122, 140, 143, 145, 187).

[JR98] Michael Jung and Ulrich Rüde. “Implicit extrapolation methods for variable coefficient problems”. In: *SIAM Journal on Scientific Computing* 19.4 (1998), pp. 1109–1124 (cit. on pp. 153, 155).

[Knu98] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. USA: Addison Wesley Longman Publishing Co., Inc., 1998. ISBN: 0201896850 (cit. on pp. 56, 164).

[KS21] Katharina Kormann and Eric Sonnendrücker. “Energy-conserving time propagation for a geometric particle-in-cell Vlasov-Maxwell solver”. In: *J. Comput. Phys.* 425 (2021), p. 109890 (cit. on p. 151).

[KKR22] Martin J Kühn, Carola Kruse, and Ulrich Rüde. “Implicitly extrapolated geometric multigrid on disk-like domains for the gyrokinetic Poisson equation from fusion plasma applications”. In: *Journal of Scientific Computing* 91.1 (2022), pp. 1–27 (cit. on pp. 8, 10, 33, 147, 152–155).

[KKR21] Martin Joachim Kühn, Carola Kruse, and Ulrich Rüde. “Energy-Minimizing, Symmetric discretisations for Anisotropic Meshes and Energy Functional Extrapolation”. In: *SIAM Journal on Scientific Computing* 43.4 (2021), A2448–A2473 (cit. on pp. 152, 153).

[Küh+21] Martin Joachim Kühn, Philippe Leleux, Carola Kruse, et al. *Gmgpolar/iexmg: complexity analysis*. to be published. 2021 (cit. on pp. 152, 155).

[Lap+08] Xavier Lapillonne, T. Dannert, Stephan Brunner, et al. “Effects of geometry on linear and non-linear gyrokinetic simulations, and development of a global version of the GENE code”. In: 1069 (Nov. 2008). DOI: 10.1063/1.3033716 (cit. on p. 122).

[Lat+07] G. Latu, N. Crouseilles, V. Grandgirard, et al. “Gyrokinetic Semi-lagrangian Parallel Simulation Using a Hybrid OpenMP/MPI Programming”. In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Ed. by Franck Cappello, Thomas Herault, and Jack Dongarra. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 356–364. ISBN: 978-3-540-75416-9 (cit. on p. 119).

[Lat11] Guillaume Latu. “Fine-Grained Parallelization of a Vlasov-Poisson Application on GPU”. In: *Euro-Par 2010 Parallel Processing Workshops*. Ed. by Mario R. Guarracino, Frédéric Vivien, Jesper Larsson Träff, et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 127–135. ISBN: 978-3-642-21878-1 (cit. on p. 89).

[LBS96] H. Lütjens, A. Bondeson, and O. Sauter. "The CHEASE code for toroidal MHD equilibria". In: *Computer Physics Communications* 97.3 (1996), pp. 219–260. ISSN: 0010-4655. DOI: https://doi.org/10.1016/0010-4655(96)00046-X. URL: https://www.sciencedirect.com/science/article/pii/001046559600046X (cit. on pp. 33, 122).

[MHD10] G Manfredi, S Hirstoaga, and S Devaux. "Vlasov modelling of parallel transport in a tokamak scrape-off layer". In: *Plasma Physics and Controlled Fusion* 53.1 (Nov. 2010), p. 015012. DOI: 10.1088/0741-3335/53/1/015012. URL: https://doi.org/10.1088/0741-3335/53/1/015012 (cit. on pp. 72, 81, 82).

[MV04] G. Manfredi and F. Valsaque. "Vlasov simulations of plasma-wall interactions in a weakly collisional plasma". In: *Computer Physics Communications* 164.1 (2004). Proceedings of the 18th International Conferene on the Numerical Simulation of Plasmas, pp. 262–268. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2004.06.037. URL: https://www.sciencedirect.com/science/article/pii/S0010465504002905 (cit. on pp. 72, 82).

[Mar+09] Alessandro Marinoni, Stephan Brunner, Y. Camenen, et al. "The effect of plasma triangularity on turbulent transport: Modeling TCV experiments by linear and non-linear gyrokinetic simulations". In: *Plasma Physics and Controlled Fusion* 51 (Mar. 2009), p. 055016. DOI: 10.1088/0741-3335/51/5/055016 (cit. on p. 122).

[Meh+13] Mehrenberger, M., Steiner, C., Marradi, L., et al. "Vlasov on GPU". In: *ESAIM: Proc.* 43 (2013), pp. 37–58. DOI: 10.1051/proc/201343003. URL: https://doi.org/10.1051/proc/201343003 (cit. on p. 89).

[Mic+21] Dominik Michels, Andreas Stegmeir, Philipp Ulbl, et al. "GENE-X: A full-f gyrokinetic turbulence code based on the flux-coordinate independent approach". In: *Computer Physics Communications* 264 (2021), p. 107986. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2021.107986. URL: https://www.sciencedirect.com/science/article/pii/S0010465521000989 (cit. on p. 145).

[Mik+18] D. R. Mikkelsen, N. T. Howard, A. E. White, et al. "Verification of GENE and GYRO with L-mode and I-mode plasmas in Alcator C-Mod". In: *Physics of Plasmas* 25.4 (2018), p. 042505. DOI: 10.1063/1.5018741. eprint: https://doi.org/10.1063/1.5018741. URL: https://doi.org/10.1063/1.5018741 (cit. on p. 122).

[Mil+98] R. L. Miller, M. S. Chu, J. M. Greene, et al. "Noncircular, finite aspect ratio, local equilibrium model". In: *Physics of Plasmas* 5.4 (1998), pp. 973–978. DOI: 10.1063/1.872666. eprint: https://doi.org/10.1063/1.872666. URL: https://doi.org/10.1063/1.872666 (cit. on pp. 122, 123).

[Mun+22] Yann Munschy, Emily Bourne, Guilhem Dif-Pradalier, et al. “Kinetic plasma-wall interaction using immersed boundary conditions”. in preparation. 2022 (cit. on pp. 5, 32, 93, 96, 187).

[Par+14] A. Paredes, H. Bufferand, G. Ciraolo, et al. “A penalization technique to model plasma facing components in a tokamak with temperature variations”. In: *Journal of Computational Physics* 274 (2014), pp. 283–298. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2014.05.025. URL: https://www.sciencedirect.com/science/article/pii/S0021999114003714 (cit. on p. 77).

[PDR07] J. Parvizian, A. Düster, and E. Rank. “Finite cell method”. In: *Comput Mech* 41 (2007), pp. 121–133. DOI: https://doi.org/10.1007/s00466-007-0173-y (cit. on p. 156).

[PM02] Nicholas M. Patrikalakis and Takashi Maekawa. “Representation of Curves and Surfaces”. In: *Shape Interrogation for Computer Aided Design and Manufacturing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 1–33. ISBN: 978-3-642-04074-0. DOI: 10.1007/978-3-642-04074-0_1. URL: https://doi.org/10.1007/978-3-642-04074-0_1 (cit. on p. 35).

[PT96] Les Piegl and Wayne Tiller. *The NURBS book*. Springer Science & Business Media, 1996 (cit. on pp. 35, 58).

[Riv+17] Fabio Riva, Emmanuel Lanti, Sébastien Jolliet, et al. “Plasma shaping effects on tokamak scrape-off layer turbulence”. In: *Plasma Physics and Controlled Fusion* 59.3 (Jan. 2017), p. 035001. DOI: 10.1088/1361-6587/aa5322. URL: https://doi.org/10.1088/1361-6587/aa5322 (cit. on p. 122).

[RH98] M. N. Rosenbluth and F. L. Hinton. “Poloidal Flow Driven by Ion-Temperature-Gradient Turbulence in Tokamaks”. In: *Phys. Rev. Lett.* 80 (4 Jan. 1998), pp. 724–727. DOI: 10.1103/PhysRevLett.80.724. URL: https://link.aps.org/doi/10.1103/PhysRevLett.80.724 (cit. on pp. 138, 187).

[Sar+11] Y. Sarazin, V. Grandgirard, J. Abiteboul, et al. “Predictions on heat transport and plasma rotation from global gyrokinetic simulations”. In: *Nuclear Fusion* 51.10 (Sept. 2011), p. 103023. DOI: 10.1088/0029-5515/51/10/103023. URL: https://doi.org/10.1088/0029-5515/51/10/103023 (cit. on p. 76).

[Sau16] O. Sauter. “Geometric formulas for system codes including the effect of negative triangularity”. In: *Fusion Engineering and Design* 112 (2016), pp. 633–645. ISSN: 0920-3796. DOI: https://doi.org/10.1016/j.fusengdes.2016.04.033. URL: https://www.sciencedirect.com/science/article/pii/S0920379616303234 (cit. on p. 126).

[Sch64] Isaac Jacob Schoenberg. “Spline interpolation and best quadrature formulae”. In: *Bulletin of the American Mathematical Society* 70.1 (1964), pp. 143–148. DOI: bams/1183525790. URL: https://doi.org/ (cit. on pp. 62, 186).

[SV67] Martin H Schultz and Richard S Varga. “L-splines”. In: *Numerische Mathematik* 10.4 (1967), pp. 345–369 (cit. on p. 108).

[Sch21] Christina Schwarz. “Geometric multigrid for the Gyrokinetic Poisson equation from fusion plasma applications”. Universität Erlangen-Nürnberg. https://elib.dlr.de/146684/. MA thesis. Universität Erlangen-Nürnberg, 2021. URL: https://elib.dlr.de/146684/ (cit. on p. 153).

[Sec65] Don Secrest. “Best Approximate Integration Formulas and Best Error Bounds”. In: *j-MATH-COMPUT* 19.89 (Apr. 1965), pp. 79–83. ISSN: 0025-5718 (print), 1088-6842 (electronic) (cit. on pp. 62, 65).

[SeL18] the SeLaLib Development Team. *Semi-Lagrangian library*. https://github.com/selalib/selalib. 2018 (cit. on pp. 8, 10, 33, 136, 170).

[Shk60] N. Shklov. “Simpson’s Rule for Unequally Spaced Ordinates”. In: *The American Mathematical Monthly* 67.10 (1960), pp. 1022–1023. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/2309244 (visited on 07/28/2022) (cit. on p. 66).

[Soc16a] Taha Sochi. “Introduction to tensor calculus”. In: *arXiv preprint arXiv:1603.01660* (2016) (cit. on p. 127).

[Soc16b] Taha Sochi. “Tensor calculus”. In: *arXiv preprint arXiv:1610.04347* (2016) (cit. on p. 127).

[Son+99] Eric Sonnendrücker, Jean Roche, Pierre Bertrand, et al. “The Semi-Lagrangian Method for the Numerical Resolution of the Vlasov Equation”. In: *Journal of Computational Physics* 149.2 (1999), pp. 201–220. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1998.6148. URL: https://www.sciencedirect.com/science/article/pii/S0021999198961484 (cit. on pp. 81, 101, 102, 114).

[SC91] Andrew Staniforth and Jean Côté. “Semi-Lagrangian integration schemes for atmospheric models—A review”. In: *Monthly weather review* 119.9 (1991), pp. 2206–2223 (cit. on p. 101).

[Str68] Gilbert Strang. “On the Construction and Comparison of Difference Schemes”. In: *SIAM Journal on Numerical Analysis* 5.3 (Sept. 1968), pp. 506–517. DOI: 10.1137/0705041 (cit. on pp. 31, 80, 113).

[SW06] H. Sugama and T.-H. Watanabe. “Collisionless damping of geodesic acoustic modes”. In: *Journal of Plasma Physics* 72.6 (2006), pp. 825–828. DOI: 10.1017/S0022377806004958 (cit. on pp. 139, 140, 187).

[Tos+17] Deepesh Toshniwal, Hendrik Speleers, René R. Hiemstra, et al. "Multi-degree smooth polar splines: A framework for geometric modeling and isogeometric analysis". In: *Computer Methods in Applied Mechanics and Engineering* 316 (2017). Special Issue on Isogeometric Analysis: Progress and Challenges, pp. 1005–1061. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2016.11.009. URL: https://www.sciencedirect.com/science/article/pii/S004578251631533X (cit. on pp. 35, 40, 43–45, 147).

[TOS00] Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. *Multigrid*. Elsevier, 2000 (cit. on pp. 152, 154).

[Var+22a] R Varennes, X Garbet, L. Vermare, et al. "Impact of magnetic ripple on neoclassical equilibrium in gyrokinetic simulations". working paper or preprint. Apr. 2022. URL: https://hal.archives-ouvertes.fr/hal-03631218 (cit. on p. 5).

[Var+22b] R. Varennes, X. Garbet, L. Vermare, et al. "Synergy of Turbulent Momentum Drive and Magnetic Braking". In: *Phys. Rev. Lett.* 128 (25 June 2022), p. 255002. DOI: 10.1103/PhysRevLett.128.255002. URL: https://link.aps.org/doi/10.1103/PhysRevLett.128.255002 (cit. on p. 5).

[VBH92] A. Vermeulen, R. Bartels, and Glenn Heppler. "Integrating Products of B-Splines". In: *Siam Journal on Scientific and Statistical Computing* 13 (July 1992). DOI: 10.1137/0913060 (cit. on p. 83).

[XC06] Yong Xiao and Peter Catto. "Plasma shaping effects on the collisionless residual zonal flow level". In: *Physics of Plasmas* 13 (Aug. 2006), pp. 082307–082307. DOI: 10.1063/1.2266892 (cit. on pp. 141, 142, 187).

[Zha+19] Weiqun Zhang, Ann Almgren, Vince Beckner, et al. "AMReX: a framework for block-structured adaptive mesh refinement". In: *Journal of Open Source Software* 4.37 (2019), pp. 1370–1370 (cit. on pp. 165, 167, 170, 176).

[Zon19] Edoardo Zoni. "Theoretical and numerical studies of gyrokinetic models for shaped Tokamak plasmas". PhD thesis. Technische Universität München, 2019 (cit. on p. 159).

[ZG19] Edoardo Zoni and Yaman Güçlü. "Solving hyperbolic-elliptic problems on singular mapped disk-like domains with the method of characteristics and spline finite elements". In: *Journal of Computational Physics* 398 (2019), p. 108889 (cit. on pp. 8, 10, 33, 136, 147, 148, 159, 160).

ANNEXES

A. Expected Conservation Error

When calculating the error for the conservation equations (3.33) - (3.35) in Chapter 3 some of the error will be due to the truncation of the domain and is therefore unavoidable. In this appendix this unavoidable error is quantified. The distribution function $f_s(t,x,v_s)$ can be approximated by a Maxwellian with density $n_s(t,x) = 1$, and $T_s(t,x) = 1$:

$$f_s(t,x,v_s) = \exp\left(-\frac{v_s^2}{2}\right) \tag{A.1}$$

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

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

$$C_i = \int_{-\infty}^{\infty} v_s^i \exp\left(-\frac{v_s^2}{2}\right) dv_s \tag{A.3}$$

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

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

$$\varepsilon = \sqrt{2\pi}\,\mathrm{erf}\left(\frac{\infty}{\sqrt{2}}\right) - \sqrt{2\pi}\,\mathrm{erf}\left(\frac{a}{\sqrt{2}}\right) = \sqrt{2\pi}\left(1 - \mathrm{erf}\left(\frac{a}{\sqrt{2}}\right)\right) \tag{A.4}$$

Table A.1 shows the truncation error for different cut-off values v_T.

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

Table A.1.: v_T necessary to attain an expected error assuming that the distribution function is Maxwellian, as defined in equation (A.4).

B. Determination of Culham Equations

The Culham equilibrium presented in Section 6.6 is defined by several parameters. The definition of these parameters can be derived from a few assumptions. In this annex I show the derivation of these definitions.

A. Magnetic Field Dependencies

The magnetic field $B(r,\theta)$ is orthogonal to ∇r which allows us to write $B(r,\theta)$ as:

$$B(r,\theta) = B_0 R_0 \left(f(r,\theta)\nabla\theta + g(r,\theta)\nabla\phi \right) \tag{B.1}$$

where $f(r,\theta)$ and $g(r,\theta)$ are functions describing the poloidal and toroidal components of the magnetic field. The divergence of $B(r,\theta)$ is 0:

$$\nabla \cdot B(r,\theta) = \nabla \cdot \left(f(r,\theta)\nabla\theta + g(r,\theta)\nabla\phi \right) = \frac{\partial f(r,\theta)}{\partial\theta} + \frac{\partial g(r,\theta)}{\partial\phi} = \frac{\partial f(r,\theta)}{\partial\theta} = 0 \tag{B.2}$$

Therefore $f(r,\theta) = f(r)$. For an axisymmetric equilibrium the magnetic field $B(r)$ can be written:

$$B(r) = I(r)\nabla\phi + \nabla\phi \times \nabla\psi(r) \tag{B.3}$$

where $I(r)$ is the plasma current and $\psi(r)$ is the flux of the magnetic field across a closed curve. Therefore we have:

$$f(r) = \frac{\psi'(r)}{B_0 R_0} \tag{B.4}$$

$$g(r) = \frac{I(r)}{B_0 R_0} \tag{B.5}$$

The functions describing $\psi(r)$ and $I(r)$ are unknown, so this definition does not allow us to calculate the values of $f(r)$ and $g(r)$.

B. Quasi-Toroidal Assumption

For the next steps we will first need some properties of the Jacobian determinant J which will be calculated in Appendix C.

$$J = \partial_r R \partial_\theta Z - \partial_r Z \partial_\theta R \tag{B.6}$$

$$\frac{1}{J} = \partial_R r \partial_Z \theta - \partial_Z r \partial_R \theta = (\nabla r \times \nabla \theta) \cdot \hat{e}_\phi = g_{\phi\phi}(\nabla r \times \nabla \theta) \cdot \nabla \phi \tag{B.7}$$

$$= R(\nabla \phi \times \nabla r) \cdot \nabla \theta = \frac{R\psi'(r)}{B_0 R_0 f(r)} (\nabla \phi \times \nabla r) \cdot \nabla \theta \tag{B.8}$$

$$= \frac{R}{B_0 R_0 f(r)} (\nabla \phi \times \nabla \psi) \cdot \nabla \theta = \frac{RB \cdot \nabla \theta}{R_0 B_0 f(r)} \tag{B.9}$$

Before proceeding, we introduce the non-physical intrinsic angle $\theta^\star$ such that:

$$\frac{B \cdot \nabla \theta^\star}{B \cdot \nabla \phi} = \frac{1}{q(r)} \tag{B.10}$$

This implies:

$$\frac{\partial \theta^\star}{\partial \theta} \frac{B \cdot \nabla \theta}{B \cdot \nabla \phi} = \frac{1}{q(r)} \tag{B.11}$$

$$\frac{\partial \theta^\star}{\partial \theta} \frac{\frac{R_0 B_0 f(r)}{R(r,\theta) J(r,\theta)}}{R_0 B_0 g(r,\theta) (g^{\phi\phi})^2} = \frac{1}{q(r)} \tag{B.12}$$

$$\frac{\partial \theta^\star}{\partial \theta} \frac{f(r) R^2}{R(r,\theta) J(r,\theta) g(r,\theta)} = \frac{1}{q(r)} \tag{B.13}$$

$$\frac{\partial \theta^\star}{\partial \theta} = \frac{J(r,\theta) g(r,\theta)}{q(r) f(r) R(r,\theta)} \tag{B.14}$$

We suppose that the transformation to the variables $(r, \theta^\star, \phi)$ is quasi-toroidal, i.e. that the Jacobian determinant $J^\star(r,\theta)$ is equal to $\frac{R(r,\theta) r}{R_0}$:

$$\frac{R_0}{R(r,\theta) r} = \frac{1}{J^\star(r,\theta)} = R(r,\theta)(\nabla r \times \nabla \theta^\star) \cdot \nabla \phi = \frac{\partial \theta^\star}{\partial \theta} R(\nabla r \times \nabla \theta) \cdot \nabla \phi \tag{B.15}$$

$$= \frac{J(r,\theta) g(r,\theta)}{q(r) f(r) R} \frac{1}{J(r,\theta)} = \frac{g(r,\theta)}{q(r) f(r) R(r,\theta)} \tag{B.16}$$

To provide an expression for this in terms of the elements of the geometry we return to the definition of the intrinsic angle. By adding the periodic boundary conditions

$\theta^\star(r,0) = \theta^\star(r,2\pi) = 0$ we obtain:

$$\theta^\star(r,\theta) = \int_0^\theta \frac{J(r,\theta')g(r)}{f(r)R(r,\theta')q(r)}d\theta' \tag{B.17}$$

$$\int_0^{2\pi} \frac{J(r,\theta')g(r)}{f(r)R(r,\theta')q(r)}d\theta' = \frac{g(r)}{f(r)q(r)}\int_0^{2\pi}\frac{J(r,\theta')}{R(r,\theta')}d\theta' = 2\pi \tag{B.18}$$

We must now calculate the integral. The expression will be truncated according to the small parameter $\varepsilon = \frac{a}{R_0}$. The harmonic terms will be truncated at $r\varepsilon$, while the non-harmonic terms will be truncated at $r\varepsilon^2$. This is equivalent to calculating an average of the terms accurate to $r\varepsilon^2$. The terms at $r\varepsilon$ are kept as multiplications of harmonic terms can lead to non-harmonic terms. Remember that $E(r)$, $T(r)$, and $\Delta(r)$ are considered to be of order $r\varepsilon^2$, while $A(r)$ is of order $r\varepsilon^3$. We first consider the truncated expression for $J(r,\theta)$:

$$\begin{aligned} J(r,\theta) = r - A(r) - rA'(r) - E(r)E'(r) - 2T(r)T'(r) + r\Delta'(r)\cos(\theta) \\ + [E(r) - rE'(r)]\cos(2\theta) + [rT'(r) - 2T(r)]\cos(3\theta) + \mathcal{O}(\epsilon^2) \end{aligned} \tag{B.19}$$

and $\frac{R}{R_0}$:

$$\frac{R(r,\theta)}{R_0} = 1 + \underbrace{\frac{\Delta(r)}{R_0}}_{\varepsilon^2} + \underbrace{\frac{r}{R_0}}_{\varepsilon}\cos(\theta) + \mathcal{O}(\epsilon^2) \tag{B.20}$$

Using the assumption that $\varepsilon \ll 0$ we therefore have:

$$\left(\frac{R(r,\theta)}{R_0}\right)^{-1} = 1 - \frac{\Delta(r)}{R_0} - \frac{r}{R_0}\cos(\theta) + \left(\frac{r}{R_0}\cos(\theta)\right)^2 + \mathcal{O}(\epsilon^2) \tag{B.21}$$

$$= 1 - \frac{\Delta(r)}{R_0} - \frac{r}{R_0}\cos(\theta) + \frac{r^2}{R_0^2}\frac{\cos(2\theta)+1}{2} + \mathcal{O}(\epsilon^2) \tag{B.22}$$

$$= 1 - \frac{\Delta(r)}{R_0} - \frac{r}{R_0}\cos(\theta) + \frac{r^2}{2R_0^2} + \mathcal{O}(\epsilon^2) \tag{B.23}$$

Which finally gives:

$$\begin{aligned} \frac{R_0 J(r,\theta)}{R(r,\theta)} =& r - A(r) - rA'(r) - E(r)E'(r) - 2T(r)T'(r) + r\Delta'(r)\cos(\theta) \\ &+ [E(r) - rE'(r)]\cos(2\theta) + [rT'(r) - 2T(r)]\cos(3\theta) \\ &- \frac{r\Delta(r)}{R_0} - \frac{r}{R_0}\cos(\theta)\left[r + r\Delta'(r)\cos(\theta) + [E(r) - rE'(r)]\cos(2\theta)\right. \\ &\left. + [rT'(r) - 2T(r)]\cos(3\theta)\right] + \frac{r^3}{2R_0^2} + \mathcal{O}(\epsilon^2) \end{aligned}$$

$$
\begin{aligned}
&= r - A(r) - rA'(r) - E(r)E'(r) - 2T(r)T'(r) + r\Delta'(r)\cos(\theta) \\
&\quad + [E(r) - rE'(r)]\cos(2\theta) + [rT'(r) - 2T(r)]\cos(3\theta) - \frac{r\Delta(r)}{R_0} - \frac{r^2}{R_0}\cos(\theta) \\
&\quad - \frac{r^2\Delta'(r)}{R_0}\frac{\cos(2\theta)+1}{2} - \frac{r}{R_0}[E(r) - rE'(r)]\frac{\cos(3\theta)+\cos(\theta)}{2} \\
&\quad - \frac{r}{R_0}[rT'(r) - 2T(r)]\frac{\cos(4\theta)+\cos(2\theta)}{2} + \frac{r^3}{2R_0^2} + \mathcal{O}(\epsilon^2) \\
&= r - A(r) - rA'(r) - E(r)E'(r) - 2T(r)T'(r) + r\Delta'(r)\cos(\theta) \\
&\quad + [E(r) - rE'(r)]\cos(2\theta) + [rT'(r) - 2T(r)]\cos(3\theta) \\
&\quad - \frac{r\Delta(r)}{R_0} - \frac{r^2}{R_0}\cos(\theta) - \frac{r^2\Delta'(r)}{2R_0} + \frac{r^3}{2R_0^2} + \mathcal{O}(\epsilon^2) \\
&\approx r - A(r) - rA'(r) - E(r)E'(r) - 2T(r)T'(r) - \frac{r\Delta(r)}{R_0} - \frac{r^2\Delta'(r)}{2R_0} + \frac{r^3}{2R_0^2} \\
&\quad + \left(r\Delta'(r) - \frac{r^2}{R_0}\right)\cos(\theta) + [E(r) - rE'(r)]\cos(2\theta) + [rT'(r) - 2T(r)]\cos(3\theta)
\end{aligned}
\tag{B.24}
$$

Returning to equation (B.18) we then have:

$$
\begin{aligned}
R_0\frac{f(r)q(r)}{g(r)} &= \frac{R_0}{2\pi}\int_0^{2\pi}\frac{J(r,\theta')}{R(r,\theta')}d\theta' \\
&\approx r - A(r) - rA'(r) - E(r)E'(r) - 2T(r)T'(r) - \frac{r\Delta(r)}{R_0} - \frac{r^2\Delta'(r)}{2R_0} + \frac{r^3}{2R_0^2}
\end{aligned}
\tag{B.25}
$$

Let us call this term γ.

This equation can be inserted into the quasi-toroidal assumption (equation (B.16)) to give:

$$
\frac{R_0}{R(r,\theta)r} = \frac{g(r,\theta)}{q(r)f(r)R(r,\theta)} = \frac{R_0}{R(r,\theta)\gamma} \tag{B.26}
$$

Therefore $\gamma = r$. We use this condition to define $A(r)$:

$$
\frac{\partial(rA(r))}{\partial r} = -E(r)E'(r) - 2T(r)T'(r) - \frac{r\Delta(r)}{R_0} + \frac{r^2\Delta'(r)}{2R_0} + \frac{r^3}{2R_0^2} \tag{B.27}
$$

$$
= \frac{\partial}{\partial r}\left(-\frac{E(r)^2}{2} - T(r)^2 - \frac{r^2\Delta(r)}{2R_0} + \frac{r^4}{8R_0^2}\right) \tag{B.28}
$$

$$
A(r) = \frac{r^3}{8R_0^2} - \frac{E(r)^2}{2r} - \frac{T(r)^2}{r} - \frac{r\Delta(r)}{2R_0} \tag{B.29}
$$

C. Geometric Parameter Definitions

The functions $f(r)$ and $g(r)$ must satisfy the Grad-Shafranov equation:

$$\nabla\cdot\nabla\psi = -\mu_0 R^2 \frac{\partial p(\psi)}{\partial \psi} - I(\psi)\frac{\partial I(\psi)}{\partial \psi} \tag{B.30}$$

$$\nabla\cdot\nabla\psi = -\mu_0 R^2 \frac{1}{\psi'(r)}\frac{\partial p(r)}{\partial r} - I(r)\frac{1}{\psi'(r)}\frac{\partial I(r)}{\partial r} \tag{B.31}$$

$$\nabla\cdot\nabla\psi(r) = -\mu_0 R^2 \frac{1}{B_0 R_0 f(r)}\frac{\partial p(r)}{\partial r} - B_0 R_0 g(r)\frac{1}{f(r)}\frac{\partial g(r)}{\partial r} \tag{B.32}$$

Using the definition of the Laplacien from Section 5.2.2.2 (equation (5.71), using the fact that there is no coefficient $\alpha = 1$, or $\beta = 0$, and that $\psi(r)$ only depends on r):

$$\begin{aligned} &-\frac{1}{J(r,\theta)}\frac{\partial}{\partial r}\left(J(r,\theta)|\nabla r|^2\frac{\partial\psi}{\partial r}\right) - \frac{1}{J(r,\theta)}\frac{\partial}{\partial\theta}\left(J(r,\theta)\nabla r\cdot\nabla\theta\frac{\partial\psi}{\partial r}\right) \\ &\quad = \mu_0 R^2\frac{1}{B_0R_0f(r)}\frac{\partial p(r)}{\partial r} + B_0R_0g(r)\frac{1}{f(r)}\frac{\partial g(r)}{\partial r} \end{aligned} \tag{B.33}$$

In order to simplify this equation we will write it in the coordinates $(r,\theta^\star,\phi)$:

$$\begin{aligned} &-\frac{R_0}{Rr}\frac{\partial}{\partial r}\left(\frac{Rr}{R_0}|\nabla r|^2\frac{\partial\psi}{\partial r}\right) - \frac{R_0}{Rr}\frac{\partial}{\partial\theta^\star}\left(\frac{Rr}{R_0}\nabla r\cdot\nabla\theta^\star\frac{\partial\psi}{\partial r}\right) \\ &\quad = \mu_0 R^2\frac{1}{B_0R_0f(r)}\frac{\partial p(r)}{\partial r} + B_0R_0g(r)\frac{1}{f(r)}\frac{\partial g(r)}{\partial r} \end{aligned} \tag{B.34}$$

$$\begin{aligned} &-\frac{1}{r}\frac{\partial}{\partial r}\left(r|\nabla r|^2\frac{\partial\psi}{\partial r}\right) - \frac{1}{r}\frac{\partial}{\partial\theta^\star}\left(r\nabla r\cdot\nabla\theta^\star\frac{\partial\psi}{\partial r}\right) \\ &\quad = \mu_0 R^2\frac{1}{B_0R_0f(r)}\frac{\partial p(r)}{\partial r} + B_0R_0g(r)\frac{1}{f(r)}\frac{\partial g(r)}{\partial r} \end{aligned} \tag{B.35}$$

$$\begin{aligned} &-\frac{1}{r}\frac{\partial}{\partial r}\left(r|\nabla r|^2 f(r)\right) - \frac{1}{r}\frac{\partial}{\partial\theta^\star}\left(r\nabla r\cdot\nabla\theta^\star f(r)\right) \\ &\quad = \mu_0\frac{R^2}{R_0^2}\frac{1}{B_0^2 f(r)}\frac{\partial p(r)}{\partial r} + g(r)\frac{1}{f(r)}\frac{\partial g(r)}{\partial r} \end{aligned} \tag{B.36}$$

To use this expression we first need to calculate $|\nabla r|^2$ and $\nabla r\cdot\nabla\theta^\star$ in the coordinates $(r,\theta^\star,\phi)$. Only terms up to the first order are required.

C.1. $|\nabla r|^2$

The expression for $|\nabla r|^2$ in the coordinate (r,θ,ϕ) is shown in Appendix C, equation (C.16). This equation and others that will be used later require an expression for the squared Jacobian which will be derived in Appendix C. Here I provide the truncated

expression:

$$\begin{aligned} J^2(r,\theta) =& \left(r + r\Delta'(r)\cos(\theta) + \left[E(r) - rE'(r)\right]\cos(2\theta) + \left[rT'(r) - 2T(r)\right]\cos(3\theta)\right)^2 \\ &+ \mathcal{O}(\varepsilon^2) \\ \approx& r^2\left(1 + \Delta'(r)\cos(\theta) + \left[\frac{E(r)}{r} - E'(r)\right]\cos(2\theta) + \left[T'(r) - \frac{2T(r)}{r}\right]\cos(3\theta)\right)^2 \\ =& r^2\left(1 + 2\Delta'(r)\cos(\theta) + 2\left[\frac{E(r)}{r} - E'(r)\right]\cos(2\theta) + 2\left[T'(r) - \frac{2T(r)}{r}\right]\cos(3\theta)\right) \end{aligned} \tag{B.37}$$

which leads to the following approximation:

$$|\nabla r|^2 = 1 - 2\Delta'(r)\cos(\theta) - 2\left[\frac{E(r)}{r} - E'(r)\right]\cos(2\theta) - 2\left[T'(r) - \frac{2T(r)}{r}\right]\cos(3\theta) + \mathcal{O}(\varepsilon^2) \tag{B.38}$$

In order to have this equation in the coordinates $(r,\theta^\star,\phi)$ we return to our definition of $\theta^\star$:

$$\theta^\star(r,\theta) = \int_0^\theta \frac{J(r,\theta)R_0}{R(r,\theta)\gamma}d\theta' = \int_0^\theta \frac{J(r,\theta)R_0}{R(r,\theta)r}d\theta' \tag{B.39}$$

Combining equations (B.24) and (B.29) we therefore obtain:

$$\begin{aligned} \frac{R_0 J(r,\theta)}{R(r,\theta)r} \approx& 1 - \left(\frac{r^2}{8R_0^2} - \frac{E(r)^2}{2r^2} - \frac{T(r)^2}{r^2} - \frac{\Delta(r)}{2R_0}\right) \\ &- \left(\frac{3r^2}{8R_0^2} - \frac{E(r)E'(r)}{r} + \frac{E(r)^2}{2r^2} - \frac{2T(r)T'(r)}{r} + \frac{T(r)^2}{r^2} - \frac{\Delta(r)}{2R_0} - \frac{r\Delta'(r)}{2R_0}\right) \\ &- \frac{E(r)E'(r)}{r} - \frac{2T(r)T'(r)}{r} - \frac{\Delta(r)}{R_0} - \frac{r\Delta'(r)}{2R_0} + \frac{r^2}{2R_0^2} \\ &+ \left(\Delta'(r) + \frac{r}{R_0}\right)\cos(\theta) + \left(\frac{E(r)}{r} - E'(r)\right)\cos(2\theta) + \left(T'(r) - \frac{2T(r)}{r}\right)\cos(3\theta) \\ \approx& 1 + \left(\Delta'(r) - \frac{r}{R_0}\right)\cos(\theta) + \left(\frac{E(r)}{r} - E'(r)\right)\cos(2\theta) + \left(T'(r) - \frac{2T(r)}{r}\right)\cos(3\theta) \\ \theta^\star \approx& \theta + \left(\Delta'(r) - \frac{r}{R_0}\right)\sin(\theta) + \left(\frac{E(r)}{r} - E'(r)\right)\frac{\sin(2\theta)}{2} + \left(T'(r) - \frac{2T(r)}{r}\right)\frac{\sin(3\theta)}{3} \end{aligned} \tag{B.40}$$

In equation (B.40) all harmonic terms are of order 1, therefore we can deduce the following equation:

$$\begin{aligned} \theta =& \theta^\star - \left(\Delta'(r) - \frac{r}{R_0}\right)\sin(\theta^\star) - \left(\frac{E(r)}{r} - E'(r)\right)\frac{\sin(2\theta^\star)}{2} \\ &- \left(T'(r) - \frac{2T(r)}{r}\right)\frac{\sin(3\theta^\star)}{3} + \mathcal{O}(\varepsilon^2) \end{aligned} \tag{B.41}$$

Which gives:

$$\begin{aligned}\cos(\theta) = &\cos(\theta^\star) - \sin(\theta^\star)\left(-\left(\Delta'(r) - \frac{r}{R_0}\right)\sin(\theta^\star) - \left(\frac{E(r)}{r} - E'(r)\right)\frac{\sin(2\theta^\star)}{2}\right.\\ &\left. - \left(T'(r) - \frac{2T(r)}{r}\right)\frac{\sin(3\theta^\star)}{3}\right)\\ = &\cos(\theta) + \frac{1}{2}\left(\Delta'(r) - \frac{r}{R_0}\right) + \mathcal{O}(\varepsilon^2)\end{aligned}$$

Proceeding similarly for the other harmonics we get:

$$\cos(2\theta) \approx \cos(2\theta^\star) + \frac{1}{2}\left(\frac{E(r)}{r} - E'(r)\right) \tag{B.42}$$

$$\cos(3\theta) \approx \cos(3\theta^\star) + \frac{1}{2}\left(T'(r) - \frac{2T(r)}{r}\right) \tag{B.43}$$

A similar treatment for the sinus functions gives:

$$\sin(\theta) = \sin(\theta^\star) \tag{B.44}$$

$$\sin(2\theta) = \sin(2\theta^\star) \tag{B.45}$$

$$\sin(3\theta) = \sin(3\theta^\star) \tag{B.46}$$

The truncated $|\nabla r|^2$, in the coordinates $(r, \theta^\star, \phi)$, is finally written as:

$$\begin{aligned}|\nabla r|^2 = &1 - 2\Delta'(r)\cos(\theta^\star) - 2\left(\frac{E(r)}{r} - E'(r)\right)\cos(2\theta^\star) - 2\left(T'(r) - \frac{2T(r)}{r}\right)\cos(3\theta^\star)\\ &+ \mathcal{O}(\varepsilon^2)\end{aligned} \tag{B.47}$$

C.2. $\nabla r \cdot \nabla \theta^\star$

We proceed similarly for $\nabla r \cdot \nabla \theta^\star$. Using equations (C.18) and (B.37)

$$r\nabla r \cdot \nabla\theta = \Delta'(r)\sin(\theta) - \left(\frac{E(r)}{r} + E'(r)\right)\sin(2\theta) + \left(\frac{2T(r)}{r} + T'(r)\right)\sin(3\theta) + \mathcal{O}(\varepsilon^2) \tag{B.48}$$

Using equation (B.40) we can then define:

$$\begin{aligned}\nabla\theta = &\nabla\theta^\star\left(1 + \mathcal{O}\left(\varepsilon\cos(\theta^\star)\right)\right) + \nabla r\frac{1}{r}\left(-\left(r\Delta''(r) - \frac{r}{R_0}\right)\sin(\theta^\star)\right.\\ &\left. + \left(rE''(r) - E'(r) + \frac{E(r)}{r}\right)\frac{\sin(2\theta^\star)}{2} - \left(rT''(r) - 2T'(r) + \frac{2T(r)}{r}\right)\frac{\sin(3\theta^\star)}{3} + \mathcal{O}(\varepsilon^2)\right)\end{aligned} \tag{B.49}$$

We therefore have:

$$
\begin{aligned}
r\nabla r\cdot\nabla\theta^\star = &\frac{\Delta'(r)\sin\left(\theta^\star\right)-\left(\frac{E(r)}{r}+E'(r)\right)\sin\left(2\theta^\star\right)+\left(\frac{2T(r)}{r}+T'(r)\right)\sin\left(3\theta^\star\right)}{1+\mathcal{O}(\varepsilon\cos(\theta^\star))}\\
&-\frac{|\nabla r|^2}{1+\mathcal{O}(\varepsilon\cos(\theta^\star))}\left(-\left(rE''(r)-\frac{r}{R_0}\right)\sin\left(\theta^\star\right)\right.\\
&+\left(rE''(r)-E'(r)+\frac{E(r)}{r}\right)\frac{\sin\left(2\theta^\star\right)}{2}\\
&\left.-\left(rT''(r)-2T'(r)+\frac{2T(r)}{r}\right)\frac{\sin\left(3\theta^\star\right)}{3}\right) \qquad \text{(B.50)}\\
\approx&\left[1+\mathcal{O}\left(\varepsilon\cos\left(\theta^\star\right)\right)\right]\left(\Delta'(r)\sin\left(\theta^\star\right)-\left(\frac{E(r)}{r}+E'(r)\right)\sin\left(2\theta^\star\right)\right.\\
&+\left(\frac{2T(r)}{r}+T'(r)\right)\sin\left(3\theta^\star\right)+\left(r\Delta''(r)-\frac{r}{R_0}\right)\sin\left(\theta^\star\right)\\
&-\left(rE''(r)-E'(r)+\frac{E(r)}{r}\right)\frac{\sin\left(2\theta^\star\right)}{2}\\
&\left.+\left(rT''(r)-2T'(r)+\frac{2T(r)}{r}\right)\frac{\sin\left(3\theta^\star\right)}{3}\right) \qquad \text{(B.51)}\\
\approx&\left(\Delta'(r)+r\Delta''(r)-\frac{r}{R_0}\right)\sin\left(\theta^\star\right)\\
&-\left(\frac{rE''(r)}{2}+\frac{E'(r)}{2}+\frac{3E(r)}{2r}\right)\sin\left(2\theta^\star\right)\\
&+\left(\frac{rT''(r)}{3}+\frac{T'(r)}{3}+\frac{8T(r)}{3r}\right)\sin\left(3\theta^\star\right) \qquad \text{(B.52)}
\end{aligned}
$$

C.3. Grad-Shafranov Approximations

We now have all the terms necessary to express equation B.36. We will do this term by term. We further suppose that the Taylor expansions of $f(r)$, $g(r)$, and $p(r)$ have the following form:

$$f = f_1 + f_3 + f_4 + \ldots \qquad \text{(B.53)}$$

$$g = 1 + g_2 + g_4 + \ldots \qquad \text{(B.54)}$$

$$p = p_0 + p_2 + p_4 + \ldots \qquad \text{(B.55)}$$

The first version of this equation arises by using the 0-th order terms of $\frac{R(r,\theta)}{R_0}$, $|\nabla r|^2$, $\nabla r\cdot\nabla\theta$, and $g(r)$:

$$\frac{1}{r}\frac{\partial}{\partial r}\left(rf(r)\right)+\mu_0\frac{1}{B_0^2 f(r)}p'(r)+\frac{g'(r)}{f(r)}=0 \qquad \text{(B.56)}$$

This equation will be useful for defining $f(r)$ and $g(r)$.

We now write equation (B.36) using only the first order terms of $\frac{R(r,\theta)}{R_0}$, $|\nabla r|^2$, $\nabla r \cdot \nabla\theta$, and $g(r)$:

$$\begin{aligned}
&\frac{1}{r}\frac{\partial}{\partial r}\left(\left(-2r\Delta'(r)\cos\left(\theta^\star\right)+2rE'(r)\cos\left(2\theta^\star\right)-2rT'(r)\cos\left(3\theta^\star\right)\right)f(r)\right)\\
+&\frac{1}{r}\frac{\partial}{\partial\theta^\star}\left(f(r)\left[\left(\Delta'(r)+r\Delta''(r)-\frac{r}{R_0}\right)\sin\left(\theta^\star\right)-\left(\frac{rE''(r)}{2}+\frac{E'(r)}{2}+\frac{3E(r)}{2r}\right)\sin\left(2\theta^\star\right)\right.\right.\\
&\left.\left.+\left(\frac{rT''(r)}{3}+\frac{T'(r)}{3}+\frac{8T(r)}{3r}\right)\sin\left(3\theta^\star\right)\right]\right)+\mu_0\frac{2r}{R_0}\frac{1}{B_0^2f(r)}p'(r)\cos(\theta)=0 \qquad \text{(B.57)}\\
&\left(-2\Delta'(r)\cos\left(\theta^\star\right)+2E'(r)\cos\left(2\theta^\star\right)-2T'(r)\cos\left(3\theta^\star\right)\right)f'(r)\\
&+\frac{f(r)}{r}\left(-(2\Delta'(r)+2r\Delta''(r))\cos\left(\theta^\star\right)+(2E'(r)+2rE''(r))\cos\left(2\theta^\star\right)\right.\\
&\left.-(2T'(r)+2rT''(r))\cos\left(3\theta^\star\right)\right)+\frac{f(r)}{r}\left[\left(\Delta'(r)+r\Delta''(r)-\frac{r}{R_0}\right)\cos\left(\theta^\star\right)\right.\\
&\left.-\left(rE''(r)+E'(r)+\frac{3E(r)}{r}\right)\cos\left(2\theta^\star\right)+\left(rT''(r)+T'(r)+\frac{8T(r)}{r}\right)\cos\left(3\theta^\star\right)\right]\\
&+\mu_0\frac{2r}{R_0}\frac{1}{B_0^2f(r)}p'(r)\cos(\theta)=0 \qquad \text{(B.58)}
\end{aligned}$$

We can rewrite this as:

$$\begin{aligned}
&\left(-2f'(r)\Delta'(r)-\frac{f(r)}{r}\Delta'(r)-f(r)\Delta''(r)-\frac{f(r)}{R_0}+\mu_0\frac{2r}{R_0}\frac{1}{B_0^2f(r)}p'(r)\right)\cos\left(\theta^\star\right)\\
&+\left(2E'(r)f'(r)+\frac{f(r)}{r}E'(r)+f(r)E''(r)-\frac{f(r)}{r}\frac{3E(r)}{r}\right)\cos\left(2\theta^\star\right)\\
&+\left(-2T'(r)f'(r)-\frac{f(r)}{r}T'(r)-f(r)T''(r)+\frac{8T(r)}{r}\right)\cos\left(3\theta^\star\right) \qquad \text{(B.59)}
\end{aligned}$$

which leaves us with the following three differential equations:

$$\Delta''(r)+\left(\frac{1}{r}+\frac{2f'(r)}{f(r)}\right)\Delta'(r)-\frac{2r\mu_0p'(r)}{R_0B_0^2f(r)^2}+\frac{1}{R_0}=0 \qquad \text{(B.60)}$$

$$E''(r)+\left(\frac{1}{r}+\frac{2f'(r)}{f(r)}\right)E'(r)-3\frac{E(r)}{r^2}=0 \qquad \text{(B.61)}$$

$$T''(r)+\left(\frac{1}{r}+\frac{2f'(r)}{f(r)}\right)T'(r)-8\frac{T(r)}{r^2}=0 \qquad \text{(B.62)}$$

Equations (B.29) (B.56) (B.60), (B.61), and (B.62)

C. Metric Tensor of the Culham geometry

The metric tensor as described in Section 5.2.1 is an important tool for defining equations in generalised coordinates. Here I present the equations necessary to calculate the metric tensor from the analytical definition of the "Culham geometry":

$$R(r,\theta) = r\cos(\theta) - E(r)\cos(\theta) + T(r)\cos(2\theta) - A(r)\cos(\theta) + \Delta(r) + R_0 \tag{C.1}$$

$$Z(r,\theta) = r\sin(\theta) + E(r)\sin(\theta) - T(r)\sin(2\theta) - A(r)\sin(\theta) \tag{C.2}$$

These equations have been coded into GYSELA in a new module "magnetic_config/culham_equil.F90".

The covariant metric tensor is defined as:

$$\begin{pmatrix} g^{rr} & g^{r\theta} \\ g^{\theta r} & g^{\theta\theta} \end{pmatrix} = \begin{pmatrix} \left((\partial_r R)^2 + (\partial_r Z)^2\right) & (\partial_r R \partial_\theta R + \partial_r Z \partial_\theta Z) \\ (\partial_r R \partial_\theta R + \partial_r Z \partial_\theta Z) & \left((\partial_\theta Z)^2 + (\partial_\theta R)^2\right) \end{pmatrix} \tag{C.3}$$

The necessary derivatives are:

$$\partial_r R = \cos(\theta) + \Delta'(r) - E'(r)\cos(\theta) + T'(r)\cos(2\theta) - A'(r)\cos(\theta), \tag{C.4}$$

$$\partial_r Z = \sin(\theta) + E'(r)\sin(\theta) - T'(r)\sin(2\theta) - A'(r)\sin(\theta), \tag{C.5}$$

$$\partial_\theta R = -r\sin(\theta) + E(r)\sin(\theta) - 2T(r)\sin(2\theta) + A(r)\sin(\theta), \tag{C.6}$$

$$\partial_\theta Z = r\cos(\theta) + E(r)\cos(\theta) - 2T(r)\cos(2\theta) - A(r)\cos(\theta). \tag{C.7}$$

C. Metric Tensor of the Culham geometry –

The covariant metric tensor elements are therefore defined as:

$$
\begin{aligned}
g^{rr} =& (\partial_r Z)^2 + (\partial_r R)^2 \qquad \text{(C.8)}\\
=& 1 - \underbrace{2A'(r) - \Delta'(r)^2 + E'(r)^2 + T'(r)^2}_{\varepsilon^2} + \underbrace{A'(r)^2}_{\varepsilon^4} \\
&+ \left[\underbrace{+2\Delta'(r)}_{\varepsilon} + \underbrace{-2\Delta'(r)E'(r) - 2E'(r)T'(r)}_{\varepsilon^2} - \underbrace{2\Delta'(r)A'(r)}_{\varepsilon^3} \right] \cos(\theta) \\
&+ \left[\underbrace{-2E'(r)}_{\varepsilon} + \underbrace{2T'(r)\Delta'(r)}_{\varepsilon^2} + \underbrace{2E'(r)A'(r)}_{\varepsilon^3} \right] \cos(2\theta) \\
&+ \left[\underbrace{2T'(r)}_{\varepsilon} - \underbrace{2T'(r)A'(r)}_{\varepsilon^3} \right] \cos(3\theta) \qquad \text{(C.9)}
\end{aligned}
$$

$$
\begin{aligned}
g^{\theta\theta} =& (\partial_\theta Z)^2 + (\partial_\theta R)^2 \qquad \text{(C.10)}\\
=& r^2 + \underbrace{E(r)^2 + 4T(r)^2 - 2rA(r)}_{r^2\varepsilon^2} + \underbrace{A(r)^2}_{r^2\varepsilon^4} \\
&- \underbrace{4E(r)T(r)}_{r^2\varepsilon^2}\cos(\theta) + \left[\underbrace{2rE(r)}_{r^2\varepsilon} - \underbrace{2E(r)A(r)}_{r^2\varepsilon^3} \right] \cos(2\theta) \\
&+ \left[\underbrace{-4rT(r)}_{r^2\varepsilon} + \underbrace{4T(r)A(r)}_{r^2\varepsilon^3} \right] \cos(3\theta) \qquad \text{(C.11)}
\end{aligned}
$$

$$
\begin{aligned}
g^{r\theta} =& \partial_r R \partial_\theta R + \partial_r Z \partial_\theta Z \qquad \text{(C.12)}\\
=& \left[\underbrace{-r\Delta'(r)}_{r\varepsilon} + \underbrace{E(r)T'(r) - 2T(r)E'(r) + E(r)\Delta'(r)}_{r\varepsilon^2} + \underbrace{\Delta'(r)A(r)}_{r\varepsilon^3} \right] \sin(\theta) \\
&+ \left[\underbrace{E(r) + rE'(r)}_{r\varepsilon} - \underbrace{2\Delta'(r)T(r)}_{r\varepsilon^2} - \underbrace{A(r)E'(r) - E(r)A'(r)}_{r\varepsilon^3} \right] \sin(2\theta) \\
&+ \left[\underbrace{-2T(r) - rT'(r)}_{r\varepsilon} + \underbrace{A(r)T'(r) + 2T(r)A'(r)}_{r\varepsilon^3} \right] \sin(3\theta) \qquad \text{(C.13)}
\end{aligned}
$$

The Jacobian determinant J is defined as:

$$
\begin{aligned}
J &= \partial_r R\,\partial_\theta Z - \partial_r Z\,\partial_\theta R \\
&= r - \underbrace{A(r) - E(r)E'(r) - 2T(r)T'(r) - rA'(r)}_{r\varepsilon^2} + \underbrace{AA'(r)}_{r\varepsilon^4} \\
&\quad + \left[\underbrace{r\Delta'(r)}_{r\varepsilon} + \underbrace{2T(r)E'(r) + E(r)\Delta'(r) + E(r)T'(r)}_{r\varepsilon^2} - \underbrace{A(r)\Delta'(r)}_{r\varepsilon^3}\right]\cos(\theta) \\
&\quad + \left[\underbrace{E(r) - rE'(r)}_{r\varepsilon} - \underbrace{2T(r)\Delta'(r)}_{r\varepsilon^2} + \underbrace{A(r)E'(r) - E(r)A'(r)}_{r\varepsilon^3}\right]\cos(2\theta) \\
&\quad + \left[\underbrace{rT'(r) - 2T(r)}_{r\varepsilon} - \underbrace{A(r)T'(r) + 2T(r)A'(r)}_{r\varepsilon^3}\right]\cos(3\theta) \qquad \text{(C.14)} \\
&\approx r - A(r) - rA'(r) - E(r)E'(r) - 2T(r)T'(r) + r\Delta'(r)\cos(\theta) \\
&\quad + [E(r) - rE'(r)]\cos(2\theta) + [rT'(r) - 2T(r)]\cos(3\theta) + \mathcal{O}(\epsilon^2) \qquad \text{(C.15)}
\end{aligned}
$$

Finally the contravariant metric tensor can be summarised as:

$$g_{rr} = |\nabla r|^2 = \frac{(\partial_\theta Z)^2 + (\partial_\theta R)^2}{J^2} = \frac{C.11}{(C.14)^2} \qquad \text{(C.16)}$$

$$g_{\theta\theta} = r^2|\nabla\theta|^2 = \frac{r^2\left[(\partial_r Z)^2 + (\partial_r R)^2\right]}{J^2} = \frac{C.9}{(C.14)^2} \qquad \text{(C.17)}$$

$$g_{r\theta} = r\nabla\theta\cdot\nabla r = -\frac{r\left[\partial_r R\partial_\theta R + \partial_r Z\partial_\theta Z\right]}{J^2} = \frac{-C.13}{(C.14)^2} \qquad \text{(C.18)}$$

D. Analytical definition of the X Point equilibrium

$$\psi(x,y) = \frac{x^4}{8} + A\left(\frac{1}{2}x^2\ln x - \frac{x^4}{8}\right) + \sum_{i=1}^{12} c_i \psi_i(x,y) \tag{D.1}$$

where $A = -0.05$ is a constant, the twelve functions $\psi_i(x,y)$ are defined as follows:

$$\psi_1(x,y) = 1, \tag{D.2}$$
$$\psi_2(x,y) = x^2, \tag{D.3}$$
$$\psi_3(x,y) = y^2 - x^2\ln x, \tag{D.4}$$
$$\psi_4(x,y) = x^4 - 4x^2y^2, \tag{D.5}$$
$$\psi_5(x,y) = 2y^4 - 9y^2x^2 + 3x^4\ln x - 12x^2y^2\ln x, \tag{D.6}$$
$$\psi_6(x,y) = x^6 - 12x^4y^2 + 7x^2y^4, \tag{D.7}$$
$$\psi_7(x,y) = 8y^6 - 140y^4x^2 + 75y^2x^4 - 15x^6\ln x + 180x^4y^2\ln x - 120x^2y^4\ln x, \tag{D.8}$$
$$\psi_8(x,y) = y, \tag{D.9}$$
$$\psi_9(x,y) = yx^2, \tag{D.10}$$
$$\psi_{10}(x,y) = y^3 - 3yx^2\ln x, \tag{D.11}$$
$$\psi_{11}(x,y) = 3yx^4 - 4y^3x^2, \tag{D.12}$$
$$\psi_{12}(x,y) = 8y^5 - 45yx^4 - 80y^3x^2\ln x + 60yx^4\ln x, \tag{D.13}$$

and the coefficients c_i are determined from the following boundary conditions:

$$\psi(1+\varepsilon, 0) = 0 \tag{D.14}$$
$$\psi(1-\varepsilon, 0) = 0 \tag{D.15}$$
$$\psi(1-\delta\varepsilon, \kappa\varepsilon) = 0 \tag{D.16}$$
$$\psi(1-1.1\delta\varepsilon, -1.1\kappa\varepsilon) = 0 \tag{D.17}$$
$$\frac{\partial\psi}{\partial y}(1+\varepsilon, 0) = 0 \tag{D.18}$$
$$\frac{\partial\psi}{\partial y}(1-\varepsilon, 0) = 0 \tag{D.19}$$
$$\frac{\partial\psi}{\partial x}(1-\delta\varepsilon, \kappa\varepsilon) = 0 \tag{D.20}$$
$$\frac{\partial\psi}{\partial x}(1-1.1\delta\varepsilon, -1.1\kappa\varepsilon) = 0 \tag{D.21}$$

$$\frac{\partial \psi}{\partial y}(1-1.1\delta\varepsilon,-1.1\kappa\varepsilon)=0 \tag{D.22}$$

$$\frac{\partial^2 \psi}{\partial y^2}(1+\varepsilon,0)=-\frac{(1+\alpha)^2}{\varepsilon\kappa^2}\frac{\partial \psi}{\partial x}(1+\varepsilon,0) \tag{D.23}$$

$$\frac{\partial^2 \psi}{\partial y^2}(1-\varepsilon,0)=-\frac{(1-\alpha)^2}{\varepsilon\kappa^2}\frac{\partial \psi}{\partial x}(1-\varepsilon,0) \tag{D.24}$$

$$\frac{\partial^2 \psi}{\partial x^2}(1-\delta\varepsilon,\kappa\varepsilon)=\frac{\kappa}{\varepsilon\cos^2\alpha}\frac{\partial \psi}{\partial y}(1-\delta\varepsilon,\kappa\varepsilon) \tag{D.25}$$