Solver comparison for Poisson-like equations on tokamak geometries

Emily Bourne [a,*], Philippe Leleux [b,c,*], Katharina Kormann [d], Carola Kruse [c], Virginie Grandgirard [a], Yaman Güçlü [e], Martin J. Kühn [f], Ulrich Rüde [c,g], Eric Sonnendrücker [e,h], Edoardo Zoni [i]

[a] *CEA, IRFM, Saint-Paul-les-Durance, F-13108, France*
[b] *IRIT@CRCT Team, INSERM, Toulouse, France*
[c] *Parallel Algorithms Team, CERFACS (Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique), 42 Avenue Gaspard Coriolis, 31057 Toulouse, France*
[d] *Ruhr-Universität Bochum, Fakultät für Mathematik, Universitätsstr. 150, 44801 Bochum, Germany*
[e] *Max-Planck Institut für Plasmaphysik, Garching, Germany*
[f] *German Aerospace Center, Institute for Software Technology, Department of High-Performance Computing, Cologne, Germany*
[g] *Friedrich-Alexander Universität Erlangen-Nürnberg, Cauerstr. 11, 91058 Erlangen, Germany*
[h] *Technische Universität München, Garching, Germany*
[i] *Lawrence Berkeley National Laboratory, Berkeley, CA, USA*

Abstract

The solution of Poisson-like equations defined on complex geometry is required for gyrokinetic simulations, which are important for the modelling of plasma turbulence in nuclear fusion devices such as the ITER tokamak. In this paper, we compare three solvers capable of solving this problem, in terms of the accuracy of the solution, and their computational efficiency. The first, the Spline FEM solver, uses $\mathcal{C}^1$ polar splines to construct a finite elements method which solves the equation on curvilinear coordinates. The resulting linear system is solved using a conjugate gradient method. The second, the GmgPolar solver, uses a symmetric finite differences method to discretise the differential equation. The resulting linear system is solved using a tailored geometric multigrid scheme, with a combination of zebra circle and radial line smoothers, together with an implicit extrapolation scheme. The third, the Embedded Boundary solver, uses a finite volumes method on Cartesian coordinates with an embedded boundary scheme. The resulting linear system is solved using a multigrid scheme. 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. All three solvers are shown to be capable of solving the equation on a realistic non-analytical geometry. The Embedded Boundary solver is additionally used to attempt to solve an X-point geometry, highlighting the problems with concave boundaries.

Keywords: plasma simulation, Poisson equation, tokamak, finite elements, finite differences, finite volumes, multigrid scheme, conjugate gradient

1. Introduction

The construction of efficient tokamak fusion reactors represents one of the most promising potential methods for future energy production. In these reactors, plasma is produced and magnetically confined inside a torus. The current design of reactors, e.g. ITER[1] or JET[2], was obtained using empirical laws. In particular, turbulent transport in plasma is a limiting factor for the performance of energy production, and

*Corresponding author
Email addresses: emily.bourne@cea.fr (Emily Bourne), leleux.philippe0@gmail.com (Philippe Leleux)
[1] https://www.iter.org/sci/beyonditer
[2] https://ccfe.ukaea.uk/research/joint-european-torus/

its underlying physics needs to be well understood in order to obtain better designs for tokamak devices. The simulation of plasma in tokamaks involves high costs in terms of computation and memory, and requires the use of efficient High Performance Computing techniques (HPC).

The gyrokinetic framework is of particular interest in this context since it is able to capture the turbulence in the plasma, and allows the phase-space dimensions of the system to be reduced from 6 to 5 dimensions: 3 for the torus geometry and 2 for the velocity [BBG+18, GAB+16]. 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 [MSU+21] and EUTERPE [HTK+02], solve this equation in its entirety, the majority of codes including GYSELA [BBG+18], and ORB5 [JBA+07] 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 article, we are interested in the solution of the 2D gyrokinetic Poisson-like equation $Lu = f$ with homogeneous Dirichlet boundary conditions, defined as:

$$\begin{aligned} Lu = -\nabla \cdot (\alpha \nabla u) + \beta u = f \text{ in } \Omega, \\ u = 0 \text{ on } \partial\Omega, \end{aligned} \tag{1}$$

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 (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 [GAB+16], 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 2), a geometric multigrid solver operating on polar coordinates, referred to as the GmgPolar solver (Section 3), and a finite differences solver operating on Cartesian coordinates with an embedded boundary approach, referred to as the Embedded Boundary solver (Section 4). The main similarities and differences of the solvers are summarised in Table 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 1: Comparison of the main similarities and differences of the three solvers. Details about these results can be found in Sections 2-4 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 1. The Cartesian coordinates are sometimes referred to as the "physical" coordinates, while the polar or curvilinear coordinates are known as the "logical" coordinates.

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 et al. [CCH+88], the cross-section can be described

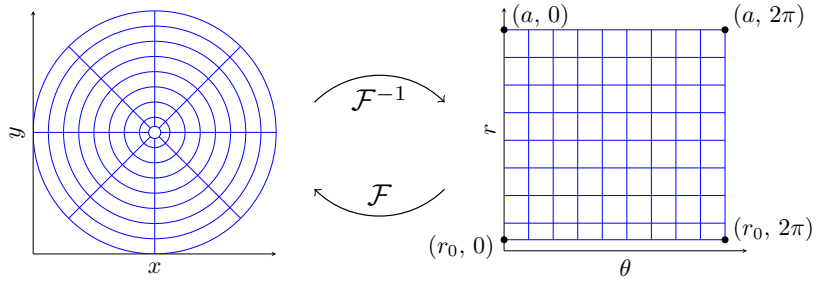


Figure 1: The curvilinear coordinates defined by a mapping $\mathcal{F}$ between the Cartesian and polar coordinates $(r, \theta) \in [r_0, a] \times [0, 2\pi)$.

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 in [CCH+88], 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 (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 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 [ZG19] to handle the singularity. The GmgPolar solver uses finite differences across the origin to avoid the issue, as detailed in [KKR22].

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 paper is organised as follows, in Section 2 the Spline FEM solver is introduced, the GmgPolar solver is introduced in Section 3, and the Embedded Boundary solver in Section 4. In Section 5, we compare these three approaches for the solution of the gyrokinetic Poisson-like equation on analytical test cases. In Section 6, we compare the behaviour of the three solvers on the non-analytical "Culham geometry". In Section 7, we discuss the difficulties encountered when tackling more complex geometries. The goal of this paper is to give the advantages of each solver with a view to an integration in the GYSELA code.

2. Spline FEM

In [ZG19], 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 et al. [TSHH17], around the singularity to ensure that a $\mathcal{C}^1$ smooth solution can be found. In this section, we summarise the scheme. For more details see [ZG19].

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 1). These break

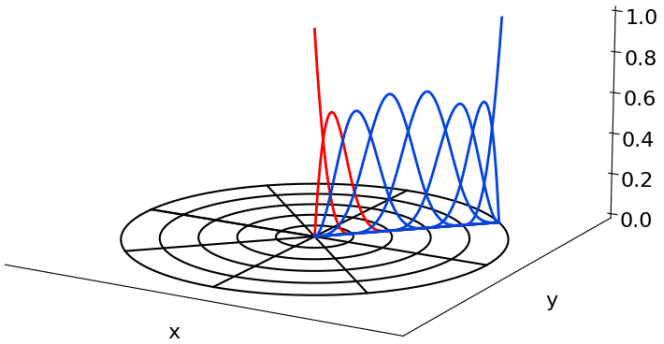


Figure 2: The radial component of the 2D basis splines of degree 3. The basis-splines which are replaced by the $\mathcal{C}^1$ polar basis functions are shown in red

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

$$k_0 = k_1 = \cdots = k_d = b_0 < k_{d+1} = b_1 < \ldots k_{n_{cr}+d} = b_{n_{cr}+1} = k_{n_{cr}+d+1} = \cdots = k_{n_{cr}+2d}, \tag{2}$$

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:

$$k_0 = b_{n_{c\theta}-d} < \cdots < k_{d-1} = b_{n_{c\theta}} < k_d = b_0 < \ldots k_{n_{c\theta}+d} = b_{n_{c\theta}} < k_{n_{c\theta}+d+1} = b_0 < \cdots < k_{n_{c\theta}+2d} = b \tag{3}$$

2D splines which do not handle the singular point are obtained using a basis defined as

$$B_l(r,\theta) = B_{in_{c\theta}+j}(r,\theta) = b_{i,d_r}(r)b_{j,d_\theta}(\theta), \tag{4}$$

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 [Far93] of the splines in the r-direction and the θ-direction.

In the solver described by Zoni and Güçlü [ZG19], 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

$$B_{in_\theta+j}(r,\theta) = b_{i,d_r}(r)b_{j,d_\theta}(r) \quad ,\forall i \in \{0,1\}, \forall 0 \leq j < n_{b\theta} \tag{5}$$

are replaced by three new basis functions $\hat{B}_l(r,\theta)$. The replaced basis functions are illustrated in Figure 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

$$\hat{B}_{l+3}(r,\theta) = B_{l+2n_{b\theta}}(r,\theta) \quad ,\forall 0 \leq l < (n_{br}-2)n_{b\theta}. \tag{6}$$

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.

With the chosen basis functions, Equation (1) can be solved using a finite elements solver by writing the equation in its weak form:

$$\int\int \left[\beta(r)u(r,\theta)\hat{B}_l(r,\theta) + \alpha(r)\nabla u(r,\theta)\cdot\nabla\hat{B}_l(r,\theta)\right]\mathcal{J}drd\theta = \int\int f(r,\theta)\hat{B}_l(r,\theta)Jdrd\theta, \tag{7}$$

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

$$u(r,\theta) = \sum_{l=0}^{n_{b\theta}(n_{br}-2)+3} u_l \hat{B}_l(r,\theta). \tag{8}$$

This allows the system to be expressed using the matrix equation

$$(M+S)\,\widehat{u} = M\widehat{f}, \tag{9}$$

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

$$M_{i,j} = \int\int \beta(r,\theta)\hat{B}_i(r,\theta)\hat{B}_j(r,\theta)\mathcal{J}\,dr\,d\theta, \tag{10}$$

and the matrix S, known as the stiffness matrix, is defined as

$$S_{i,j} = \int\int \alpha(r,\theta)\left[\sum_{\xi_1\in\{r,\theta\}}\sum_{\xi_2\in\{r,\theta\}} \frac{\partial \hat{B}_i}{\partial \xi_1}(r,\theta) g^{\xi_1\xi_2} \frac{\partial \hat{B}_j}{\partial \xi_2}(r,\theta)\right]\mathcal{J}\,dr\,d\theta. \tag{11}$$

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. The "Culham geometry" which will be considered in Section 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 $(n_{br}-2)n_{b\theta}(2\cdot d_r+1)(2d_\theta+1)+9+6n_\theta d_r = \mathcal{O}\left(n_{br}n_{b\theta}d_r d_\theta\right)$.

3. Geometric multigrid solver for curvilinear coordinates (GmgPolar)

In the design of the GmgPolar solver in [KKR21, KKR22, LSK+22], 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 [TOS00]. 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 [Bar88], and then generalised to curvilinear geometries in [KKR22]. 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 [KKR21, KKR22].

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 (1) on the curvilinear domain would lead to a non-symmetric matrix. Since symmetric matrices are numerically advantageous, we instead discretise the energy functional

$$J(u) := \int_\Omega \left(\frac{1}{2}\alpha|\nabla u|^2 + \frac{1}{2}\beta u^2 - fu\right) d(x,y), \tag{12}$$

related to Equation (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 [KKR21]. In addition to this finite differences discretisation, we include a technique called *implicit extrapolation*. This technique was introduced by Jung and Rüde in [JR98] 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 [KKR22]. For a more detailed motivation of the implicit extrapolation, see [Sch21, 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 [KKR21], 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 [KKR22], 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 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 (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 [JR98, KKR22]. 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 $R_l^{l+1} = {P_{l+1}^l}^T$ [BHM00].

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 [TOS00]. 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 [Bar88], 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.

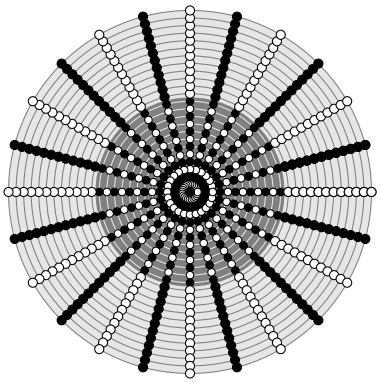


Figure 3: The circular domain is split in two subdomains (shown with light grey and dark grey background colours) depending on the use of a circle line smoother (dark grey) and a radial line smoothing (light grey). The nodes on these subdomains are coloured alternatingly in black and white. Corresponding colouring schemes are used for deformed geometries such as Figure 6a or Figure 6b.

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 [Bar88], and has been empirically shown to be the best choice also in our case, see [KKR22]. 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 3. This is useful to obtain a partial parallelisation of the combined relaxation method. 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 [KKR22]. 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[3][ADLK01], with the version 5.4.1.

Based on all of the previous elements, the multigrid scheme, possibly combined with an implicit extrapolation [JR98, KKR22] 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 [KKR22]. 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 [KKR22].
- 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 [LSK+22]. 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.

[3] http://mumps.enseeiht.fr/

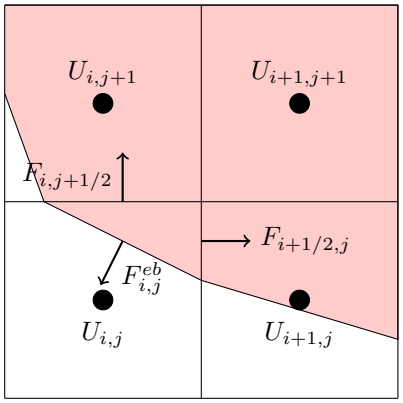


Figure 4: Illustration of the finite volume approach with embedded boundaries. The shaded part of the cells is inside, the white part outside of the domain. The fluxes through the boundary of the lower left cell are shown.

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 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 [JC98, BH12], the cut finite elements method [BCH+15], and the finite cell method [PDR07]. The AMReX library [tADTAB+22, ea19] implements a finite volume solver based on the work of Johansen and Colella [JC98]. 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 4 illustrates the situation.

For the finite volume solution, Equation (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 (1) is discretised as

$$(LU)_{i,j} = -\frac{1}{\Delta x \Delta y}\left(F_{i+1/2,j} - F_{i-1/2,j} + F_{i,j+1/2} - F_{i,j-1/2}\right) + \beta(x_i, y_j)U_{i,j}, \tag{13}$$

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

$$F_{i+1/2,j} = \Delta y\, \alpha(x_{i+1/2}, y_j)\frac{U_{i+1,j} - U_{i,j}}{\Delta x}. \tag{14}$$

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 (13) is modified on the partially covered cells to

$$(LU)_{i,j} = -\frac{1}{\Delta x \Delta y \Lambda_{i,j}}\left(F_{i+1/2,j} - F_{i-1/2,j} + F_{i,j+1/2} - F_{i,j-1/2} - F^{eb}_{i,j}\right) + \beta(x_i, y_j)U_{i,j}. \tag{15}$$

For the fraction on the cell boundary, the modified flux formula is given as

$$F_{i+1/2,j} = a_{i+1/2,j}\Delta y \alpha_m \left(\frac{1 + a_{i+1/2,j}}{2} \frac{U_{i+1,j} - U_{i,j}}{\Delta x} + \frac{1 - a_{i+1/2,j}}{2} \frac{U_{i+1,j+1} - U_{i,j+1}}{\Delta x} \right), \tag{16}$$

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 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 [JC98] 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 [ABC+98].

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 paper, 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 (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.

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 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 [GAB+16]. 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.

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 (1). In curvilinear coordinates, these coefficients depend only on the radius r. We define the coefficients similarly to Zoni [Zon19]:

$$\alpha(r) = \exp\left[-\tanh\left(\frac{r - r_p}{\delta_r}\right)\right], \tag{17}$$

$$\beta(r) = -1/\alpha(r). \tag{18}$$

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 5.

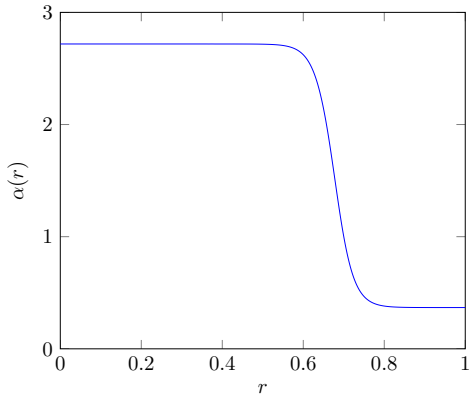


Figure 5: Radial profile of the diffusivity coefficient $\alpha(r)$ defined in (17) for the gyrokinetic Equation (1).

The first geometry is a stretched ellipse with a Shafranov shift defined by the mapping

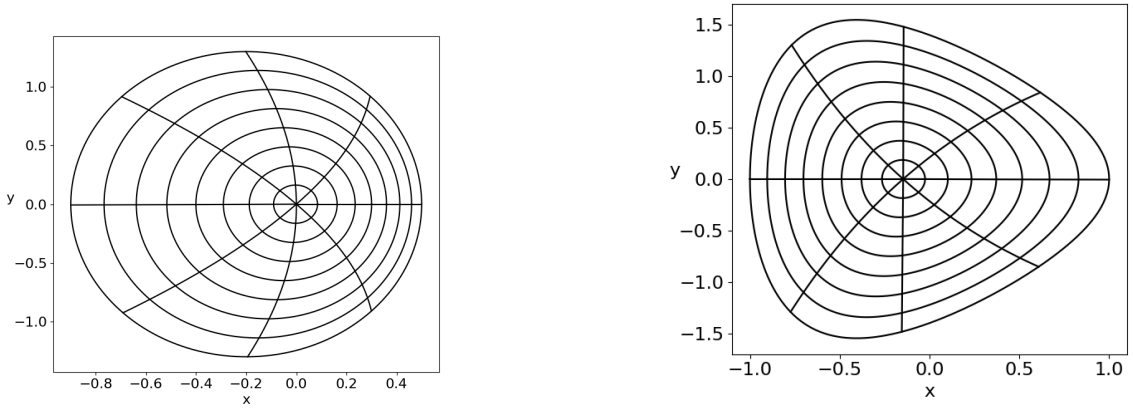
$$\begin{aligned} x(r,\theta) &= (1 - E_0) r \cos\theta - \delta_0 r^2, \\ y(r,\theta) &= (1 + E_0) r \sin\theta, \end{aligned} \tag{19}$$

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 [ZG19]: $E_0 = 0.3$, and $\delta_0 = 0.2$. The second geometry, originally proposed by Czarny et al.[CH08], has a triangular shape and ellipticity, and is defined by the mapping:

$$\begin{aligned} x(r,\theta) &= \frac{1}{\varepsilon}\left(1 - \sqrt{1 + \varepsilon\left(\varepsilon + 2r\cos\theta\right)}\right), \\ y(r,\theta) &= y_0 + \frac{e\xi r \sin\theta}{2 - \sqrt{1 + \varepsilon\left(\varepsilon + 2r\cos\theta\right)}} = y_0 + \frac{e\xi r \sin\theta}{1 + \varepsilon x(r,\theta)}, \end{aligned} \tag{20}$$

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 [ZG19]: $y_0 = 0$, $\varepsilon = 0.3$ and $e = 1.4$. Figure 6 shows the "Shafranov" and the "Czarny" geometries. These geometries have previously been investigated by Bouzat et al. [BBG+18] and Zoni and Güçlü [ZG19].

We now introduce three manufactured solutions which respect the homogeneous Dirichlet boundary conditions:



(a) "Shafranov" geometry defined by Equation (19) with $E_0 = 0.3$ and $\delta_0 = 0.2$

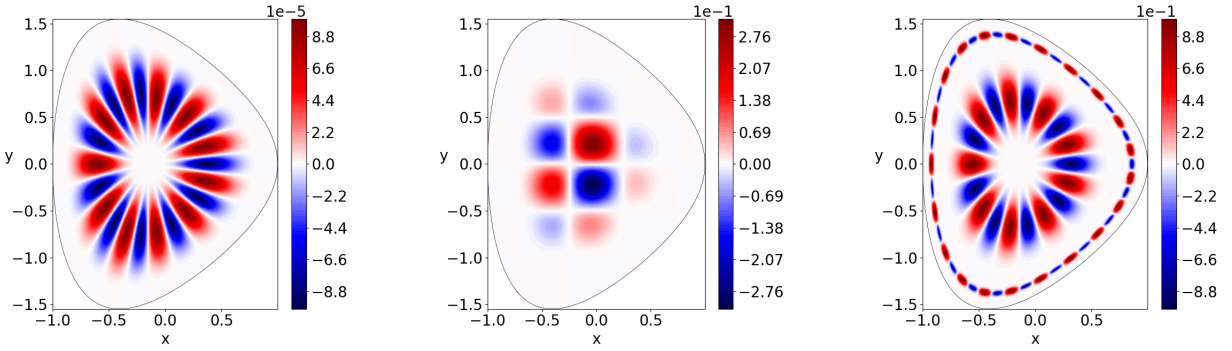
(b) "Czarny" geometry defined by Equation (20) with $y_0 = 0$, $\varepsilon = 0.3$, and $e = 1.4$

Figure 6: *Analytical geometries of the domain for the 2D gyrokinetic Poisson-like equation.*

1. **Polar solution:** A solution with oscillations aligned with the polar grid which can be used as an initial perturbation in the GYSELA code [GAB+16]:
$$u(x,y) = C(r(x,y))^6(r(x,y)-1)^6\cos(m\,\theta), \tag{21}$$
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:
$$u(x,y) = C\,(1+r(x,y))^6\,(1-r(x,y))^6\cos(2\pi x)\sin(2\pi y), \tag{22}$$
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 [DPGS+22]. The solution is defined as:
$$u(x,y) = h(r(x,y),\ 0.45,\ 0.02)\cos(9\,\theta) + h(r(x,y),\ 0.9,\ 0.0003)\cos(21\,\theta), \tag{23}$$
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:
$$\begin{aligned} h(r,c,w) =& g(r,c,w) - r\frac{\partial g}{\partial r}(0,c,w) - g(0,c,w) \\ &+ \left(r\frac{\partial g}{\partial r}(0,c,w) + g(0,c,w) - g(1,c,w)\right) r^2, \end{aligned} \tag{24}$$
$$g(r,c,w) = \exp(-(r-c)^2/w), \tag{25}$$

The right hand side f corresponding to these solutions is obtained analytically. Figure 7 shows the shape of all three solutions on the "Czarny" geometry defined in Equation (20).

We begin by considering four test cases on equidistant meshes: the polar solution (Figure 7a) and the Cartesian solution (Figure 7b) defined on both the "Shafranov" geometry (Figure 6a) and the "Czarny" geometry (Figure 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 7c).



(a) Polar solution defined in Equation (21)

(b) Cartesian solution defined in Equation (22)

(c) Multi-scale solution defined in Equation (23)

Figure 7: Shape of the manufactured solutions on the "Czarny" geometry defined by Equation (20).

5.2. Accuracy

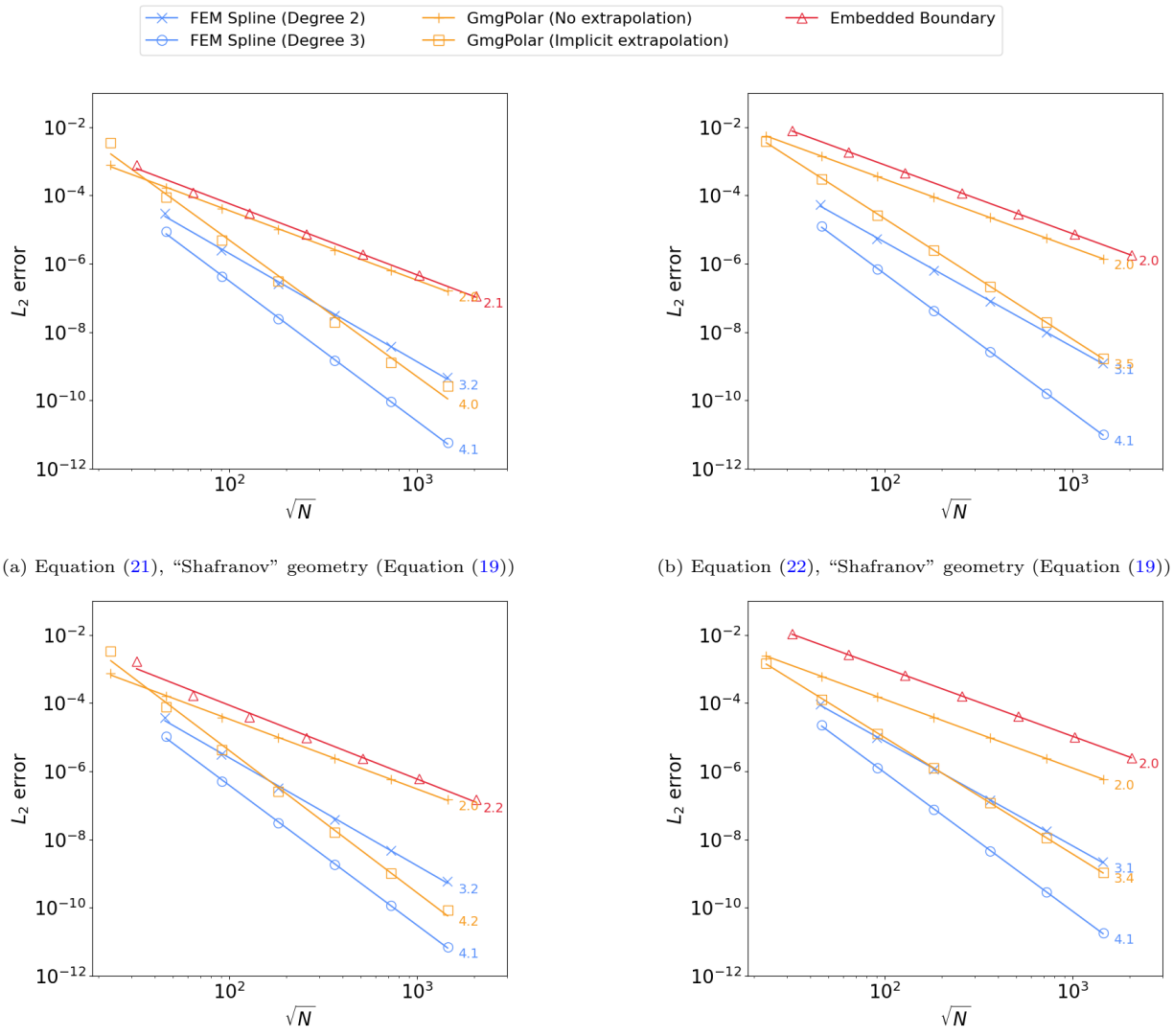
The results of the L_2-error convergence from the application of the three solvers for the four equidistant cases can be seen in Figure 8. In order to ensure that each solver has converged to the most accurate result possible, the stopping criteria is set to 10^{-14} for the Spline FEM solver, and 10^{-11} 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 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 (22), 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.

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



(a) Equation (21), "Shafranov" geometry (Equation (19))

(b) Equation (22), "Shafranov" geometry (Equation (19))

(c) Equation (21), "Czarny" geometry (Equation (20))

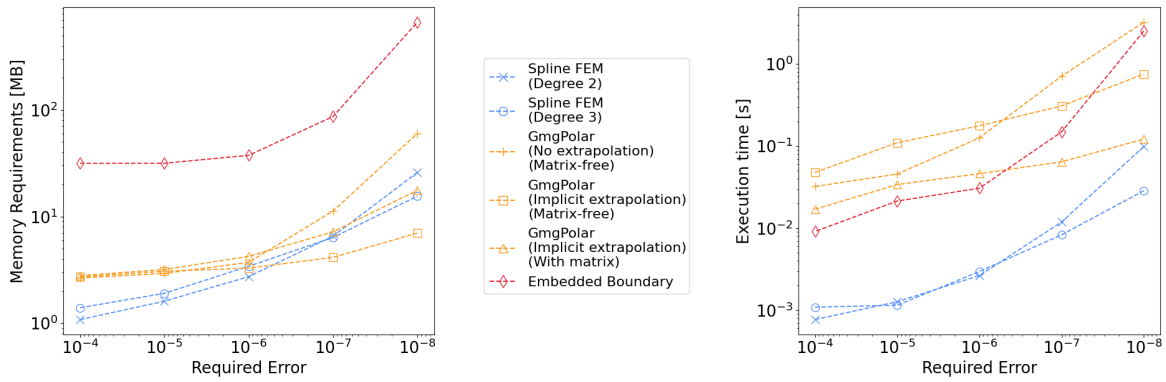
(d) Equation (22), "Czarny" geometry (Equation (20))

Figure 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.

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 [Knu98]. 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 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

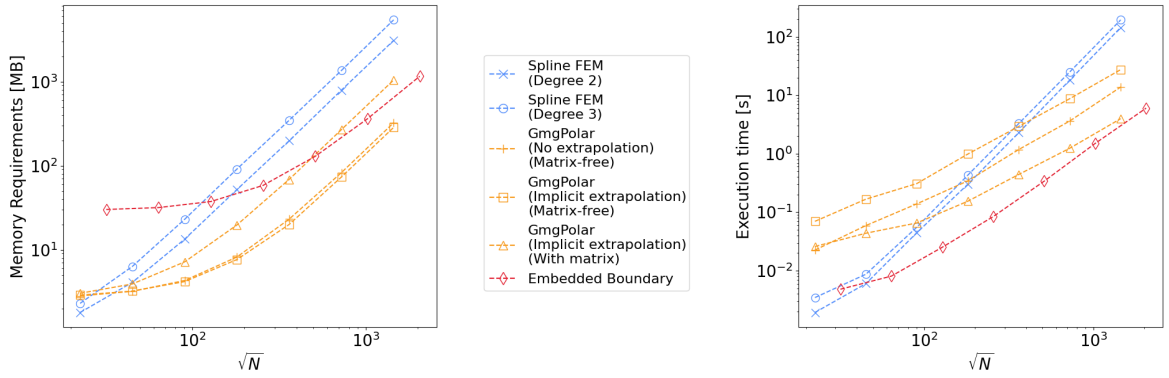
(b) Serial execution speed

Figure 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 (21) on the "Czarny" geometry described by Equation (20).

Figure 9 shows the serial performance of the different methods as a function of the required error, for the polar solution, described by Equation (21), on the "Czarny" geometry, described by Equation (20). Figure 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 solveris 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 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 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[ZAB+19]. 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.



(a) Memory requirements

(b) Serial execution speed

Figure 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 (21) on the "Czarny" geometry described by Equation (20).

In the context of a plasma simulation such as the GYSELA code [GAB+16], the error arising from the Poisson solver may not be the limiting factor for the choice of points. Other methods used in the simulation, 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 10 shows the serial performance of the different

methods for the solution described by Equation (21) on the "Czarny" geometry described by Equation (20), 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.

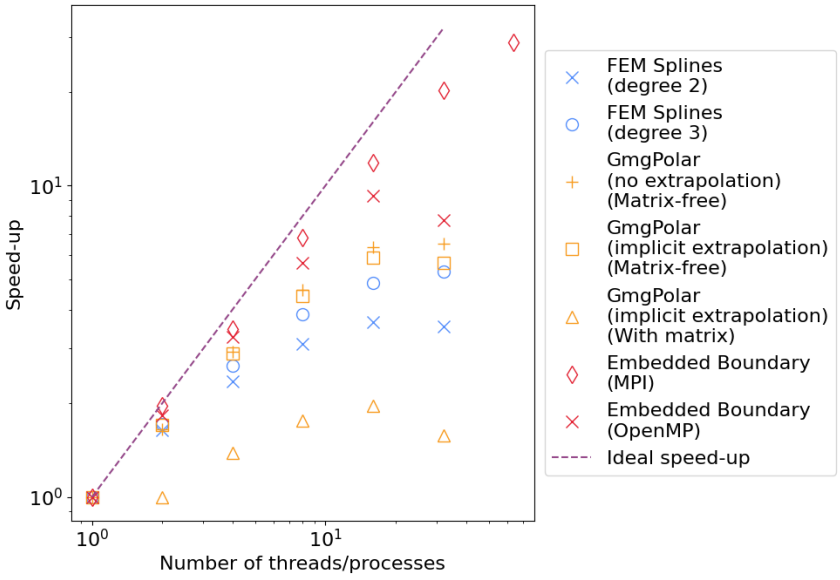


Figure 11: Execution time to solve for the polar solution described by Equation (21) on the "Czarny" geometry described by Equation (20) on 2048×2048 points.

Figure 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[ZAB+19]. 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 11 shows the performance of each solver in a parallel setup for the polar solution described by Equation (21) on the "Czarny" geometry described by Equation (20) 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.

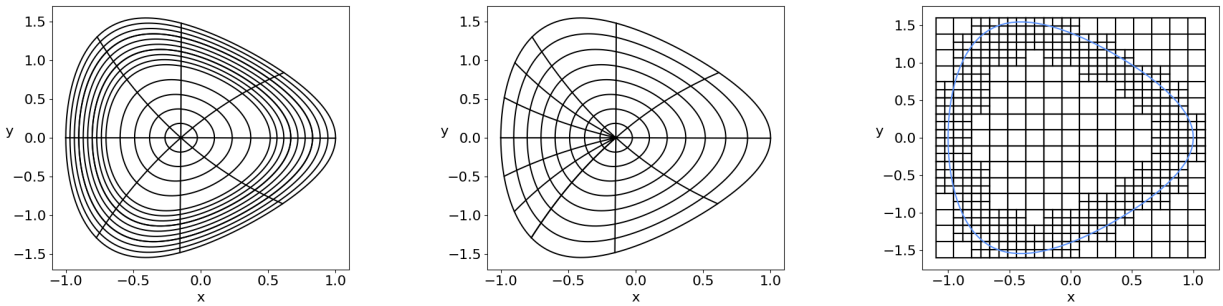
To summarise, for equidistant points Figure 8 shows that the Spline FEM solver allows us to obtain the smallest errors, and therefore the smallest number of points for a given error. Thus, Figure 9b shows that the Spline FEM solver is the fastest to attain a specific error. Then, the results from Figure 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 11. Finally, the results from Figures 9a and 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.

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 7c), i.e. a localised radius refinement as shown in Figure 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 12b.
3. on a specific patch where the error is higher, as shown in Figure 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 2

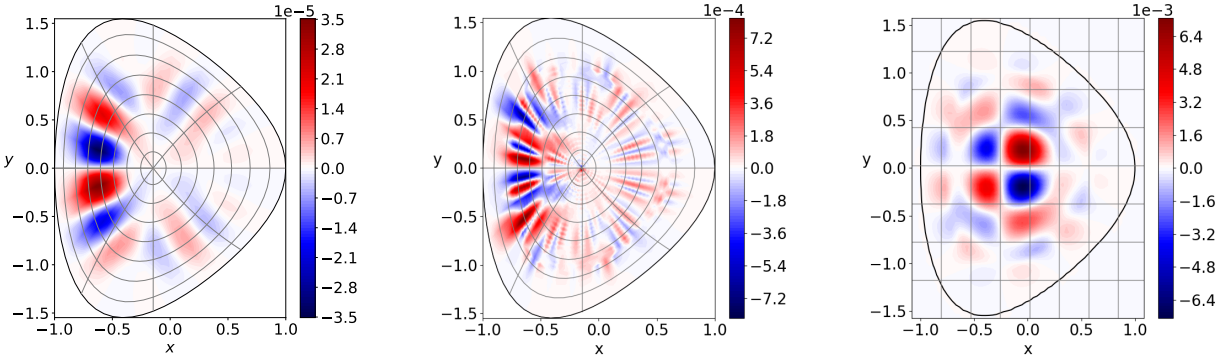


(a) Radial refinement (b) Poloidal Refinement (c) Refinement on Cartesian patches

Figure 12: Different ways to define and refine grids on the Czarny geometry.

and 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 4.

As in Section 5.2, the stopping criteria is set to 10^{-14} for the Spline FEM solver, and 10^{-11} for the other solvers.

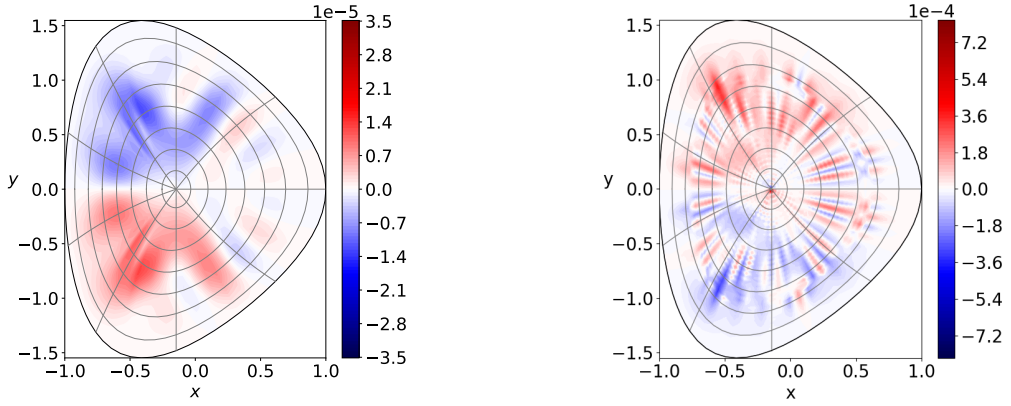


(a) Cubic Spline FEM solver, the L_∞ norm of the error is $3.51 \cdot 10^{-5}$

(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 13: The error, normalised by the ∞-norm of the solution, obtained when solving for Equation (22) on the Czarny geometry defined by (20) with 64×64 points.



(a) Cubic Spline FEM solver, the L_∞ norm of the error is $1.29 \cdot 10^{-5}$

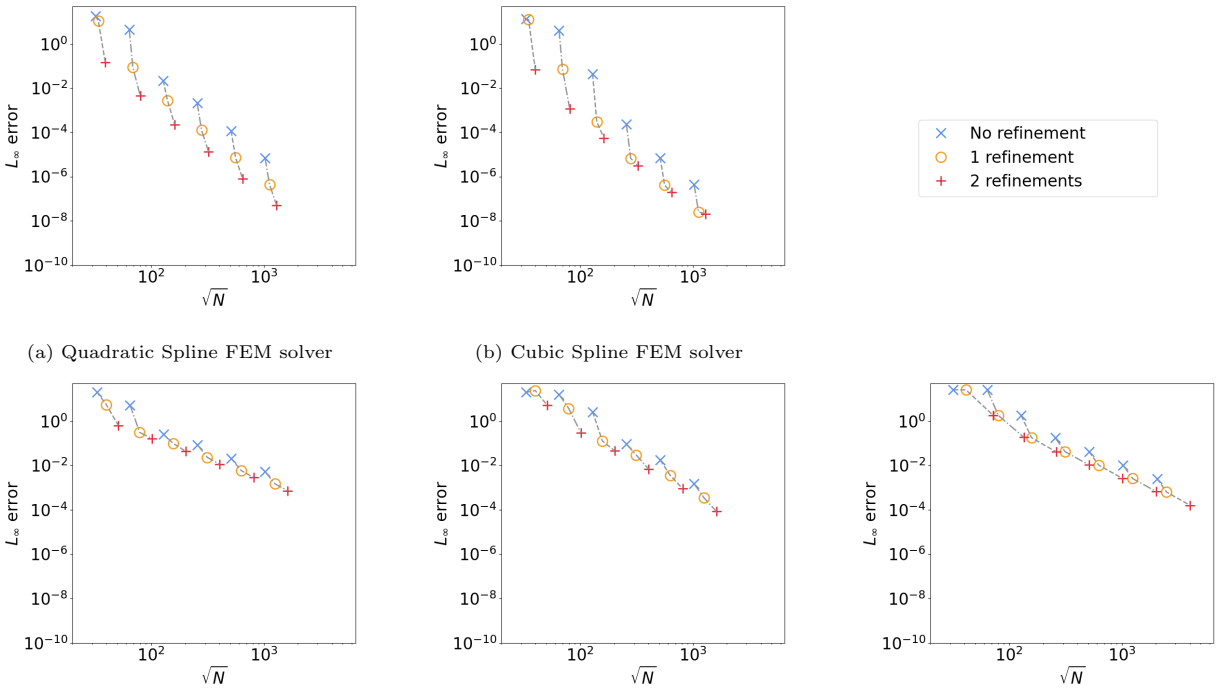
(b) GmgPolar solver, the L_∞ norm of the error is $6.91 \cdot 10^{-4}$

Figure 14: The error, normalised by the ∞-norm of the solution, obtained when solving for Equation (22) on the "Czarny" geometry defined by (20) 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 $\theta = [-\pi/4, \pi/4]$

Figure 13 shows the errors obtained when solving for Equation (22) on the "Czarny" geometry defined by Equation (20) 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 13c has a similar shape to that of the manufactured solution shown in Figure 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 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 [DPGS+22]. In order to examine this possibility, we will investigate the solution shown in Figure 7c and described by Equation (23), 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 12a. The Embedded Boundary solver will use patches overlapping this region as shown in Figure 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.



(a) Quadratic Spline FEM solver

(b) Cubic Spline FEM solver

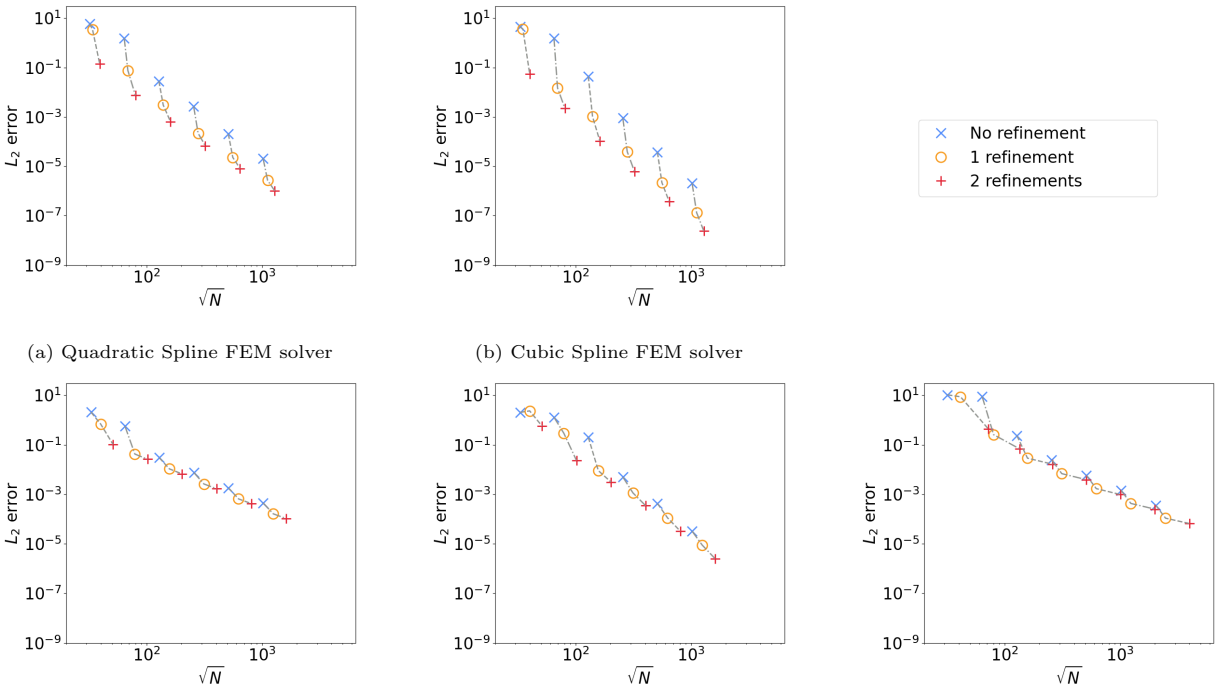
(c) GmgPolar solver without extrapolation

(d) GmgPolar solver with implicit extrapolation

(e) Embedded Boundary solver

Figure 15: L_∞ norm of the error, normalised by the ∞-norm of the solution, obtained when solving for the multi-scale solution described by Equation (23) on "Czarny" geometry described by Equation (23) 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.

Figure 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.



(a) Quadratic Spline FEM solver

(b) Cubic Spline FEM solver

(c) GmgPolar solver without extrapolation

(d) GmgPolar solver with implicit extrapolation

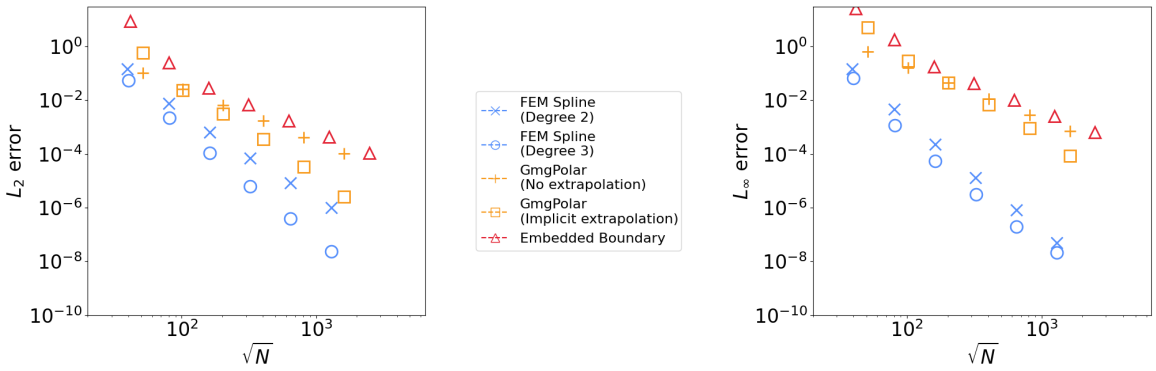
(e) Embedded Boundary solver

Figure 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 (23) on "Czarny" geometry described by Equation (23) 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.

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 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 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 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.



(a) L_2 error

(b) L_∞ error

Figure 17: Errors, normalised by the ∞-norm of the solution, when solving for Equation (23) on "Czarny" geometry defined by Equation (20) 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 $r \in [0.8, 1.0]$.

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[ZAB+19], the Spline FEM solver is based on the SeLaLib library[tSDT], and the GmgPolar solver uses the MUMPS library[ADLK01]. 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

$$n = N_C \cdot C2^{l-1}, \tag{26}$$

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. Culham Geometry

We will now apply our three solvers to the non-analytical "Culham geometry" [CCH+88]. As mentioned in Section 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:

$$\begin{aligned} x(r,\theta) &= r\cos(\theta) - E(r)\cos(\theta) + T(r)\cos(2\theta) - P(r)\cos(\theta) + \delta(r) + R_0 \\ y(r,\theta) &= r\sin(\theta) + E(r)\sin(\theta) - T(r)\sin(2\theta) - P(r)\sin(\theta) \end{aligned} \tag{27}$$

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:

$$P(r) = \frac{r^3}{8R_0^2} + \frac{r\delta(r)}{2R_0} - \frac{E(r)^2}{2r} - \frac{T(r)^2}{r}. \tag{28}$$

The geometric properties are defined as follows:

$$E''(r) + \left(\frac{1}{r} + \frac{2f'(r)}{f(r)}\right)E'(r) - 3\frac{E(r)}{r^2} = 0, \tag{29}$$

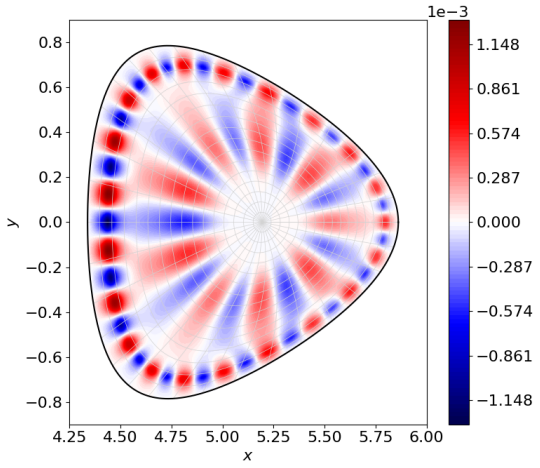
$$T''(r) + \left(\frac{1}{r} + \frac{2f'(r)}{f(r)}\right)T'(r) - 8\frac{T(r)}{r^2} = 0, \tag{30}$$

$$\delta'(r) = -\frac{1}{R_0 r^2 f(r)^2}\left(\int_0^r r' f(r')^2 dr' - \int_0^r \frac{2r'^2\mu_0 p'(r')}{B_0^2} dr'\right), \tag{31}$$

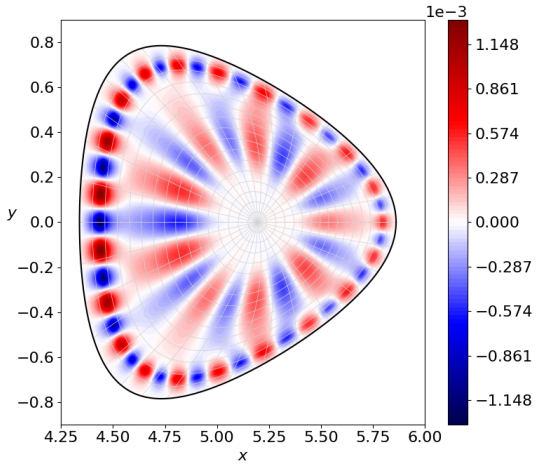
where μ_0 is the magnetic constant, $p(r)$ is the plasma pressure, and $f(r)$ and B_0 are terms used to define the magnetic field $B(r)$:

$$B(r) = B_0 R_0 \left(f(r)\hat{e}_\theta + g(r)\hat{e}_\phi\right), \tag{32}$$

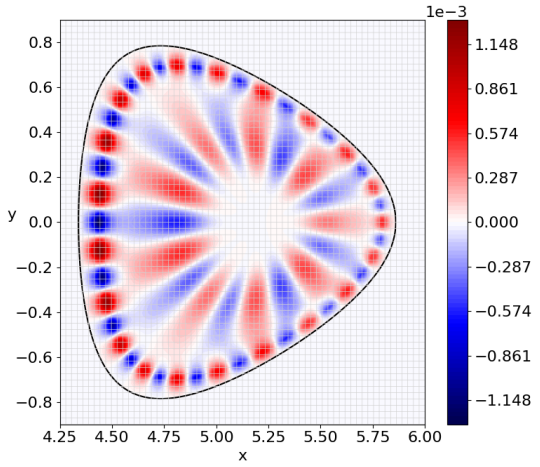
where $\hat{e}_\theta$ and $\hat{e}_\phi$ are the unit vectors in the poloidal and toroidal directions. The integration constants of the functions $E(r)$, $T(r)$ and $\delta(r)$ are defined using the constants C_E and C_T such that $E(a) = C_E$, $T(a) = C_T$, and $\delta(a) = 0$.



(a) Solution calculated on "Culham geometry" with the Spline FEM solver



(b) Solution calculated on "Culham geometry" with the Gmg-Polar solver



(c) Solution calculated on "Culham geometry" with the Embedded Boundary solver

Figure 18: The solution to Equation (1) with $\alpha(r)$ defined by Equation (17), $\beta(r) = 1/\alpha(r)$, and the right hand side $f(x, y)$ defined in the same way as the solution in the multi-scale example defined in Equation (23) on the "Culham geometry".

The functions $f(r)$ and $g(r)$ are approximated from the following system of equations:

$$f(r) = \zeta(r)g(r), \tag{33}$$

$$\zeta(r) = \frac{r}{q(r)R_0}, \tag{34}$$

$$g'(r) = \frac{\left(\frac{\zeta(r)}{r} + \zeta'(r)\right) g(r) + \frac{\mu_0 p'(r)}{B_0^2 p_0 f(r)}}{\zeta(r) + \frac{1}{f(r)}}, \tag{35}$$

where $q(r)$ is the classical safety factor in the large aspect ratio approximated by the following equation:

$$q(r) = q_0 + (q_a - q_0)r^2, \tag{36}$$

where $q_0 = q(0)$ and $q_a = q(a)$. In our investigations, we will use the following definition of the plasma pressure:

$$p(r) = p_a + (p_0 - p_a)\left(1 - r^2\right)^\gamma, \tag{37}$$

where γ is a constant, and p_0 and p_a are the pressures at respectively $r = 0$ and $r = a$. Equations (29), (30), and (35) are solved using a fourth order Runge-Kutta method with 1000 equidistant radial points and the following initial conditions:

$$E(r_0) = r_0, \tag{38}$$

$$E'(r_0) = 1, \tag{39}$$

$$T(r_0) = r_0^2, \tag{40}$$

$$T'(r_0) = 2r_0, g(r_0) = 1. \tag{41}$$

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 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 2: Parameters used to define the "Culham geometry" in Equations (27) - (37)

Figure 18 shows the result of solving Equation (1) with $\alpha(r)$ defined by Equation (17), $\beta(r) = 1/\alpha(r)$, and the right hand side $f(x, y)$ defined in the same way as the solution in the multi-scale example defined in Equation (23). 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 requiring 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.

Table 3 shows the results from using the three solvers. As in Section 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: $1.30 \cdot 10^{-3}$.

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 3: Comparison of the performance of the three solvers on the "Culham geometry" with the right hand side defined by Equation (23).

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 paper. 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 Last Closed Flux Surface (LCFS) where they no longer describe the magnetic field lines.

Figure 19 shows the results obtained using the Embedded Boundary solver with a right hand side defined by Equation (23). 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 et al. [CF10]. 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 Appendix A.

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 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 [HHP+21], 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 simulta-

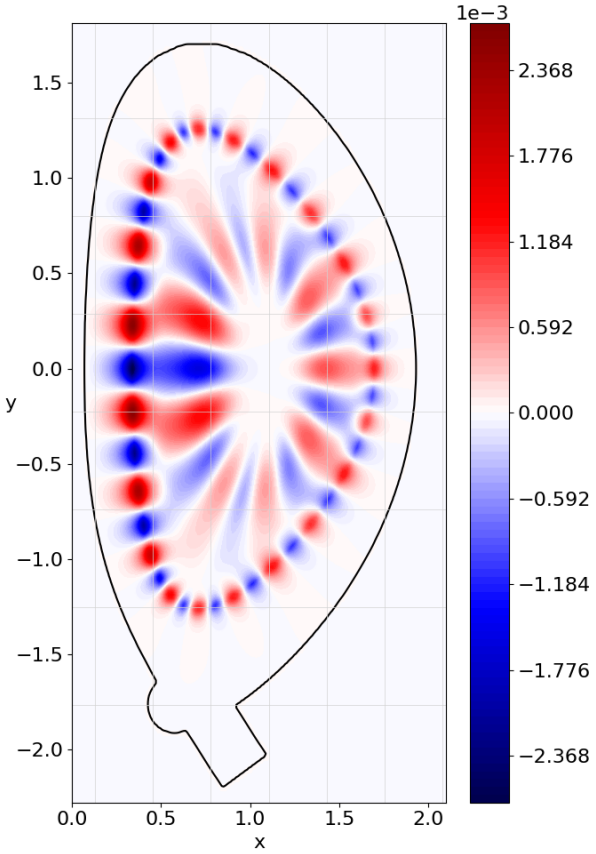


Figure 19: The solution obtained when solving Equation (1) with the right hand side defined by Equation (23) on an X-Point configuration with 256 points in the x-direction, and 256 points in the y-direction

neously. In Soledge3X [BBB+22] 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 paper.

Conclusion

We have presented three solvers for the gyrokinetic Poisson-like equation defined by Equation (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" [CCH+88], 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 [ZAB+19]). 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 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.

Acknowledgements

K.K. acknowledges advice on AMReX by Weiqun Zhang. E.B. acknowledges advice on the Culham geometry by Kevin Obrejan. Centre de Calcul Intensif d'Aix-Marseille is acknowledged for granting access to its high performance computing resources. The project has received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 824158 (EoCoE-II). 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

Appendix A. Analytical definition of the X Point equilibrium

$$\psi(x,y) = \frac{x^4}{8} + A\left(\frac{1}{2}x^2\ln x - \frac{x^4}{8}\right) + \sum_{i=1}^{12} c_i\psi_i(x,y) \tag{A.1}$$

where $A = -0.05$ is a constant, the twelve functions $\psi_i(x,y)$ are defined as follows:

$$\psi_1(x,y) = 1, \tag{A.2}$$
$$\psi_2(x,y) = x^2, \tag{A.3}$$
$$\psi_3(x,y) = y^2 - x^2\ln x, \tag{A.4}$$
$$\psi_4(x,y) = x^4 - 4x^2y^2, \tag{A.5}$$

$$\psi_5(x,y) = 2y^4 - 9y^2x^2 + 3x^4 \ln x - 12x^2y^2 \ln x, \tag{A.6}$$
$$\psi_6(x,y) = x^6 - 12x^4y^2 + 7x^2y^4, \tag{A.7}$$
$$\psi_7(x,y) = 8y^6 - 140y^4x^2 + 75y^2x^4 - 15x^6 \ln x + 180x^4y^2 \ln x - 120x^2y^4 \ln x, \tag{A.8}$$
$$\psi_8(x,y) = y, \tag{A.9}$$
$$\psi_9(x,y) = yx^2, \tag{A.10}$$
$$\psi_{10}(x,y) = y^3 - 3yx^2 \ln x, \tag{A.11}$$
$$\psi_{11}(x,y) = 3yx^4 - 4y^3x^2, \tag{A.12}$$
$$\psi_{12}(x,y) = 8y^5 - 45yx^4 - 80y^3x^2 \ln x + 60yx^4 \ln x, \tag{A.13}$$

and the coefficients c_i are determined from the following boundary conditions:

$$\psi(1+\varepsilon, 0) = 0 \tag{A.14}$$
$$\psi(1-\varepsilon, 0) = 0 \tag{A.15}$$
$$\psi(1-\delta\varepsilon, \kappa\varepsilon) = 0 \tag{A.16}$$
$$\psi(1-1.1\delta\varepsilon, -1.1\kappa\varepsilon) = 0 \tag{A.17}$$
$$\frac{\partial\psi}{\partial y}(1+\varepsilon, 0) = 0 \tag{A.18}$$
$$\frac{\partial\psi}{\partial y}(1-\varepsilon, 0) = 0 \tag{A.19}$$
$$\frac{\partial\psi}{\partial x}(1-\delta\varepsilon, \kappa\varepsilon) = 0 \tag{A.20}$$
$$\frac{\partial\psi}{\partial x}(1-1.1\delta\varepsilon, -1.1\kappa\varepsilon) = 0 \tag{A.21}$$
$$\frac{\partial\psi}{\partial y}(1-1.1\delta\varepsilon, -1.1\kappa\varepsilon) = 0 \tag{A.22}$$
$$\frac{\partial^2\psi}{\partial y^2}(1+\varepsilon, 0) = -\frac{(1+\alpha)^2}{\varepsilon\kappa^2}\frac{\partial\psi}{\partial x}(1+\varepsilon, 0) \tag{A.23}$$
$$\frac{\partial^2\psi}{\partial y^2}(1-\varepsilon, 0) = -\frac{(1-\alpha)^2}{\varepsilon\kappa^2}\frac{\partial\psi}{\partial x}(1-\varepsilon, 0) \tag{A.24}$$
$$\frac{\partial^2\psi}{\partial x^2}(1-\delta\varepsilon, \kappa\varepsilon) = \frac{\kappa}{\varepsilon\cos^2\alpha}\frac{\partial\psi}{\partial y}(1-\delta\varepsilon, \kappa\varepsilon) \tag{A.25}$$

References

[ABC+98] Ann S. Almgren, John B. Bell, Phillip Colella, Louis H. Howell, and Michael L. Welcome. A conservative adaptive projection method for the variable density incompressible Navier–Stokes equations. *Journal of Computational Physics*, 142(1):1–46, 1998.

[ADLK01] Patrick R Amestoy, Iain S Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[Bar88] Saulo RM Barros. The Poisson equation on the unit disk: a multigrid solver using polar coordinates. *Applied Mathematics and Computation*, 25(2):123–135, 1988.

[BBB+22] H Bufferand, J Balbin, S Baschetti, J Bucalossi, G Ciraolo, Ph Ghendrih, R Mao, N Rivals, P Tamain, H Yang, G Giorgiani, F Schwander, M Scotto d'Abusco, E Serre, J Denis, Y Marandet, M Raghunathan, P Innocente, and D Galassi. Implementation of multi-component Zhdanov closure in SOLEDGE3x. *Plasma Physics and Controlled Fusion*, 64(5):055001, mar 2022.

[BBG+18] Nicolas Bouzat, Camilla Bressan, Virginie Grandgirard, Guillaume Latu, and Michel Mehrenberger. Targeting realistic geometry in tokamak code gysela. *ESAIM: Proceedings and Surveys*, 63:179–207, 2018.

[BCH+15] E. Burman, S. Claus, P. Hansbo, M.G. Larson, and A. Massing. CutFEM: Discretizing geometry and partial differential equations. *International Journal for Numerical Methods in Engineering*, 104(7):472 – 501, 2015.

[BH12] Marsha Berger and Christiane Helzel. A simplified h-box method for embedded boundary grids. *SIAM Journal on Scientific Computing*, 34(2):A861–A888, 2012.

[BHM00] William L Briggs, Van Emden Henson, and Steve F McCormick. *A multigrid tutorial*. SIAM, 2000.

[CCH+88] J. W. Connor, S. C. Cowley, R. J. Hastie, T. C. Hender, A. Hood, and T. J. Martin. Tearing modes in toroidal geometry. *The Physics of Fluids*, 31(3):577–590, 1988.

[CF10] Antoine J. Cerfon and Jeffrey P. Freidberg. "one size fits all" analytic solutions to the Grad–Shafranov equation. *Physics of Plasmas*, 17(3):032502, 2010.

[CH08] Olivier Czarny and Guido Huysmans. Bézier surfaces and finite elements for MHD simulations. *Journal of Computational Physics*, 227(16):7423–7445, 2008.

[DPGS+22] G. Dif-Pradalier, Ph. Ghendrih, Y. Sarazin, E. Caschera, F. Clairet, Y. Camenen, P. Donnel, X. Garbet, V. Grandgirard, Y. Munschy, L. Vermare, and F. Widmer. Transport barrier onset and edge turbulence shortfall in fusion plasmas. *Communications Physics*, 2022. in press.

[ea19] Zhang et al. AMReX: A Framework for Block-Structured Adaptive Mesh Refinement. *Journal of Open Source Software*, 4(37):1370, 2019.

[Far93] Gerald Farin. *Curves and Surfaces for Computer-Aided Geometric Design (Third Edition)*. Academic Press, Boston, third edition edition, 1993.

[GAB+16] Virginie Grandgirard, Jérémie Abiteboul, Julien Bigot, Thomas Cartier-Michaud, Nicolas Crouseilles, Guilhem Dif-Pradalier, Ch Ehrlacher, Damien Esteve, Xavier Garbet, Ph Ghendrih, et al. A 5d gyrokinetic full-f global semi-lagrangian code for flux-driven ion turbulence simulations. *Computer Physics Communications*, 207:35–68, 2016.

[HHP+21] M. Hoelzl, G.T.A. Huijsmans, S.J.P. Pamela, M. Bécoulet, E. Nardon, F.J. Artola, B. Nkonga, C.V. Atanasiu, V. Bandaru, A. Bhole, D. Bonfiglio, A. Cathey, O. Czarny, A. Dvornova, T. Fehér, A. Fil, E. Franck, S. Futatani, M. Gruca, H. Guillard, J.W. Haverkort, I. Holod, D. Hu, S.K. Kim, S.Q. Korving, L. Kos, I. Krebs, L. Kripner, G. Latu, F. Liu, P. Merkel, D. Meshcheriakov, V. Mitterauer, S. Mochalskyy, J.A. Morales, R. Nies, N. Nikulsin, F. Orain, J. Pratt, R. Ramasamy, P. Ramet, C. Reux, K. Särkimäki, N. Schwarz, P. Singh Verma, S.F. Smith, C. Sommariva, E. Strumberger, D.C. van Vugt, M. Verbeek, E. Westerhof, F. Wieschollek, and J. Zielinski. The JOREK non-linear extended MHD code and applications to large-scale instabilities and their control in magnetically confined fusion plasmas. *Nuclear Fusion*, 61(6):065001, may 2021.

[HTK+02] Roman Hatzky, Trach Minh Tran, Axel Könies, Ralf Kleiber, and Simon J. Allfrey. Energy conservation in a nonlinear gyrokinetic particle-in-cell code for ion-temperature-gradient-driven modes in θ-pinch geometry. *Physics of Plasmas*, 9(3):898–912, 2002.

[JBA+07] S. Jolliet, A. Bottino, P. Angelino, R. Hatzky, T.M. Tran, B.F. Mcmillan, O. Sauter, K. Appert, Y. Idomura, and L. Villard. A global collisionless PIC code in magnetic coordinates. *Computer Physics Communications*, 177(5):409–425, 2007.

[JC98] Hans Johansen and Phillip Colella. A cartesian grid embedded boundary method for Poisson's equation on irregular domains. *Journal of Computational Physics*, 147(1):60–85, 1998.

[JR98] Michael Jung and Ulrich Rüde. Implicit extrapolation methods for variable coefficient problems. *SIAM Journal on Scientific Computing*, 19(4):1109–1124, 1998.

[KKR21] Martin Joachim Kühn, Carola Kruse, and Ulrich Rüde. Energy-minimizing, symmetric discretisations for anisotropic meshes and energy functional extrapolation. *SIAM Journal on Scientific Computing*, 43(4):A2448–A2473, 2021.

[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. *Journal of Scientific Computing*, 91(1):1–27, 2022.

[Knu98] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., USA, 1998.

[LSK+22] Philippe Leleux, Christina Schwarz, Martin Joachim Kühn, Carola Kruse, and Ulrich Rüde. Complexity

analysis and scalability of a matrix-free extrapolated geometric multigrid solver for curvilinear coordinates representations from plasma fusion applications. In preparation, 2022.

[MSU+21] Dominik Michels, Andreas Stegmeir, Philipp Ulbl, Denis Jarema, and Frank Jenko. Gene-x: A full-f gyrokinetic turbulence code based on the flux-coordinate independent approach. *Computer Physics Communications*, 264:107986, 2021.

[PDR07] J. Parvizian, A. Düster, and E. Rank. Finite cell method. *Comput Mech*, 41:121–133, 2007.

[Sch21] Christina Schwarz. Geometric multigrid for the Gyrokinetic Poisson equation from fusion plasma applications. Master's thesis, Universität Erlangen-Nürnberg, 2021. Universität Erlangen-Nürnberg. https://elib.dlr.de/146684/.

[tADTAB+22] the AMReX Development Team, Ann Almgren, Vince Beckner, Johannes Blaschke, Cy Chan, Marcus Day, Brian Friesen, Kevin Gott, Daniel Graves, Axel Huebl, Maximilian Katz, Andrew Myers, Tan Nguyen, Andrew Nonaka, Michele Rosso, Sam Williams, Weiqun Zhang, and Michael Zingale. Amrex-codes/amrex: Amrex 22.06, June 2022.

[TOS00] Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. *Multigrid*. Elsevier, 2000.

[tSDT] the SeLaLib Development Team. Semi-Lagrangian library. https://github.com/selalib/selalib.

[TSHH17] Deepesh Toshniwal, Hendrik Speleers, René R. Hiemstra, and Thomas J.R. Hughes. Multi-degree smooth polar splines: A framework for geometric modeling and isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 316:1005–1061, 2017. Special Issue on Isogeometric Analysis: Progress and Challenges.

[ZAB+19] Weiqun Zhang, Ann Almgren, Vince Beckner, John Bell, Johannes Blaschke, Cy Chan, Marcus Day, Brian Friesen, Kevin Gott, Daniel Graves, et al. Amrex: a framework for block-structured adaptive mesh refinement. *Journal of Open Source Software*, 4(37):1370–1370, 2019.

[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. *Journal of Computational Physics*, 398:108889, 2019.

[Zon19] Edoardo Zoni. *Theoretical and numerical studies of gyrokinetic models for shaped Tokamak plasmas*. PhD thesis, Technische Universität München, 2019.