---

**Article:**
Hill, Peter Alec orcid.org/0000-0003-3092-1858, Shanahan, Brendan William and Dudson, Benjamin Daniel orcid.org/0000-0002-0094-4867 (2016) Dirichlet boundary conditions for arbitrary-shaped boundaries in stellarator-like magnetic fields for the Flux-Coordinate Independent method. Computer Physics Communications. pp. 1-10. ISSN 0010-4655

---

# Dirichlet boundary conditions for arbitrary-shaped boundaries in stellarator-like magnetic fields for the Flux-Coordinate Independent method

Peter Hill, Brendan Shanahan, Ben Dudson

*York Plasma Institute, University of York*

**Abstract**

We present a technique for handling Dirichlet boundary conditions with the Flux Coordinate Independent (FCI) parallel derivative operator with arbitrary-shaped material geometry in general 3D magnetic fields. The FCI method constructs a finite difference scheme for $\nabla_\parallel$ by following field lines between poloidal planes and interpolating within planes. Doing so removes the need for field-aligned coordinate systems that suffer from singularities in the metric tensor at null points in the magnetic field (or equivalently, when $q \to \infty$). One cost of this method is that as the field lines are not on the mesh, they may leave the domain at any point between neighbouring planes, complicating the application of boundary conditions.

The Leg Value Fill (LVF) boundary condition scheme presented here involves an extrapolation/interpolation of the boundary value onto the field line end point. The usual finite difference scheme can then be used unmodified. We implement the LVF scheme in BOUT++ and use the Method of Manufactured Solutions to verify the implementation in a rectangular domain, and show that it doesn't modify the error scaling of the finite difference scheme. The use of LVF for arbitrary wall geometry is outlined.

We also demonstrate the feasibility of using the FCI approach in non-axisymmetric configurations for a simple diffusion model in a "straight stellarator" magnetic field. A Gaussian blob diffuses along the field lines, tracing out flux surfaces. Dirichlet boundary conditions impose a last closed flux surface (LCFS) that confines the density. Including a poloidal limiter moves the LCFS to a smaller radius.

The expected scaling of the numerical perpendicular diffusion, which is a consequence of the FCI method, in stellarator-like geometry is recovered. A novel technique for increasing the parallel resolution during post-processing, in order to reduce artefacts in visualisations, is described.

*Keywords:* plasma, stellarator

*Email address:* Peter.Hill@york.ac.uk (Peter Hill)

## 1. Introduction

Anisotropic phenomena are prevalent in magnetised plasmas. The Lorentz force tends to confine charged particles to magnetic field lines, with the result that the characteristic size of spatial variations of macroscopic plasma quantities are larger in the direction parallel to the magnetic field compared to those in the perpendicular plane.

Computational techniques take advantage of this anisotropy by, for example, aligning the computational grid to the magnetic field and reducing the resolution in the parallel direction. However, field-aligned coordinate systems typically have difficulties handling changes in magnetic topology; X-points, for instance, introduce singularities in the metric tensor. The Flux Coordinate Independent (FCI) parallel derivative operator[1–4] does not require a field-aligned coordinate system, allowing the use of simpler grids in the perpendicular plane while still allowing efficient handling of anisotropic physics.

In this work, we extend the FCI technique to handle arbitrarily shaped boundaries, including limiters, and demonstrate its use in stellarator-like fields. This work is organised as follows: in section 2, we explain the FCI method and discuss its implementation; in sections 3 and 4, we discuss some issues about interpolation and non-axisymmetric magnetic fields; simulations of stellarator-like magnetic fields are in section 5. We also describe a novel technique for upscaling visualisations in section 5.2.

## 2. Flux-Coordinate Independent method for parallel derivatives

Conventionally in magnetised plasma turbulence simulations, derivatives parallel to the magnetic field are taken by using a field-aligned coordinate system. However, these are tied to flux surfaces, and hence suffer from inevitable singularities in the metric tensor when attempting to encompass multiple magnetic topologies, i.e. crossing separatrices. These singularities can be numerically challenging to handle.

The Flux-Coordinate Independent (FCI) method for the parallel derivatives of a function is conceptually simple: one first follows the magnetic field line from a given grid point in both directions until it intersects the two adjacent perpendicular planes (see fig. 1). The function to be differentiated is then interpolated in the perpendicular plane at the field intersection points, and a finite difference scheme can be constructed using these values and the value at the emitting grid point. Higher order finite difference schemes may be constructed by following the field line past further perpendicular planes, interpolating at each intersection point. It should be noted at this point that while FCI is strictly formulated on perpendicular planes, in practice, poloidal planes are often used. This is a reasonable approximation, given the assumptions of strong anisotropy required by FCI, and we use the terms “perpendicular” and “poloidal” interchangeably throughout this work. Note that this approximation, and the identification of

perpendicular and poloidal planes, is weaker in low-aspect ratio tokamaks, and breaks down entirely in devices such as Reversed-Field Pinches.

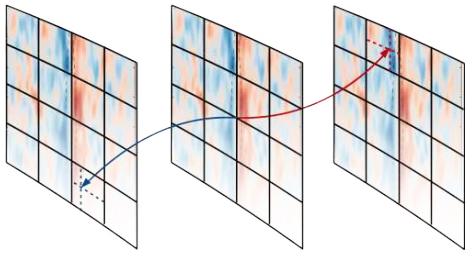


Figure 1: Schematic of the Flux Coordinate Independent method for the parallel derivative operator. Starting from a given grid point, magnetic field lines are traced in the forward and backward directions. The argument of the operator is interpolated to find the value at the location where the field line intersects the adjacent perpendicular slices, allowing a finite difference scheme to be constructed.

As the finite difference scheme is constructed at each individual grid point, the coordinate system in the perpendicular plane is no longer tied to the flux surfaces and in principle any mesh may be used. Other concerns may limit the choice of mesh, e.g. the need for easy flux-surface averages, which may require a flux-surface mesh in part of the plasma. Another consideration is that while it is possible to vastly drop the resolution in the parallel direction (i.e. the inter-plane spacing) with only a small loss in accuracy, similar to conventional field-aligned grids, one must still retain enough resolution in the perpendicular mesh to capture the relevant physics of interest.

### *2.1. Comparison with the standard* BOUT++ *mesh*

BOUT++[5–7] is a free and open source framework designed to solve partial differential equations, with an emphasis on models of magnetically confined plasmas. It has been used for a variety of applications, from edge[8–10] and scrape-off layer[11, 12] physics in tokamaks, to turbulence in linear devices[13, 14].

BOUT++ discretises space on a three-dimensional mesh, with the dimensions labelled $x$, $y$ and $z$. Typically, $x$ is the "radial" direction, $y$ the "poloidal", and $z$ the "toroidal". The conventional "ballooning"-style BOUT++ coordinate system[5, 15], for $\psi, \theta, \zeta$ the usual orthogonal tokamak coordinates, is defined as:

$$x = \psi, \qquad y = \theta, \qquad z = \zeta - \int_{\theta_0}^{\theta} \nu d\theta, \tag{1}$$

where $\nu$ is the local field line pitch, given by

$$\nu(\psi, \theta) = \frac{\partial \zeta}{\partial \theta} = \frac{\vec{B} \cdot \nabla \zeta}{\vec{B} \cdot \nabla \theta}. \tag{2}$$

By keeping $z$ fixed and moving in $y$, the integral in $z$ changes so we need to move in $\zeta$. This moves us along a field line. Essentially, $y$ is the coordinate along the field line while $z$ picks out different field lines. Because the physics of interest are expected to be field-aligned, we are able to use a lower resolution in $y$ and still resolve the physical scales.

The metric tensor for this coordinate system is orthogonal only at one $y$-location, meaning as we move in $y$, cross-terms appear in the $x$-derivatives. It is possible to eliminate these cross-terms by applying a shifted metric[1, 16]. To do this, at each $y$-point, we can shift $z$ by the integral in eq. (1), effectively moving us back into non-field-aligned coordinates, performing the derivatives in $x$, and then transforming back to the field-aligned coordinates. This can be done using Fast Fourier Transforms (FFTs) which are computationally inexpensive.

At either $y$-end of the grid we need to shift in $z$ in order to match the field lines in a twist-shift boundary[17]. This needs to be done regardless of whether or not we choose to use the shifted metric to eliminate the $x$-derivative cross-terms.

In contrast to the standard BOUT++ coordinate system, the FCI method explicitly does not use field-aligned coordinates. The construction of the parallel derivatives in fact has the major advantages of a field-aligned system (reduced resolution in the parallel direction) but allows more freedom in the choice of coordinates for the perpendicular directions. For example, two possible choices of coordinate system are tokamak coordinates:

$$x = \psi, \qquad y = \zeta, \qquad z = \theta, \tag{3}$$

or cylindrical coordinates:

$$x = R, \qquad y = \zeta, \qquad z = Z. \tag{4}$$

FCI inherently employs a shifted metric, so no cross-terms appear in the the perpendicular derivatives, simplifying the calculations, and no twist-shift has to be performed.

### *2.2. Boundary conditions*

#### *2.2.1. Simple geometry*

While FCI has already been implemented in other codes[1, 2, 4] and used for plasma simulations[3], the boundaries of the simulation domain were either periodic, or treated very simply. The problem is how to treat field lines correctly when they intersect with or leave the simulation boundaries. For example, in Ref. [2], the magnetic topology was a cylinder, and a mask was applied to the simulation domain such that the equations were not solved outside of a radius $r$. A different solution was used in Ref. [3], where the simulation was periodic in two directions, and the component of the magnetic field in the third direction was damped close to the edges, such that the resulting field was tangential to the edge. Field lines then never intersected the domain boundaries, and boundary conditions could be applied in the perpendicular direction only.

Let us first consider a scalar field $f$ on a simple, uniform, rectangular grid with boundaries located at half the grid spacing outside the first and last points in each of the grid dimensions. For any given point in the grid where the field line traced from this point intersects the boundary before intersecting the next perpendicular plane, we need to be able to calculate parallel derivatives. This situation is depicted in fig. 2, where $f_2$ is the value of the scalar field at the point in question, $f_1$ and $f_3$ are the values at the intersection points with the adjacent perpendicular planes in the negative and positive $y$ directions, respectively; $f_b$ is the value on the boundary; $l_{1,2,3}$ are the parallel distances between $f_{1,2}, f_{2,b}, f_{b,3}$ respectively.

For a Dirichlet boundary condition, we have a prescribed value on the boundary, $f_b$, which may be a function of time and/or space [1]. Given uniform spacing in $y$, we also have $l_2 + l_3 = l_1 = dy$. The question then is given $l_{1,2,3}, f_{1,2,b}$, what is $f_2'$?

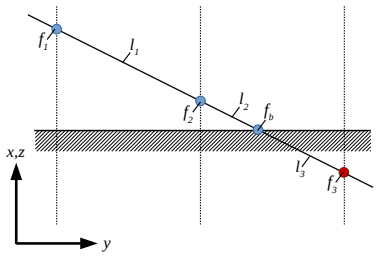



Figure 2: A field line leaving the boundary. $f_2$ is located on a grid point, while $f_1, f_3$ are located on intersection points with the adjacent perpendicular planes, and $f_b$ is located on the intersection with the boundary. $l_1, l_2, l_3$ are distances along the field lines between the four points above.

We should like to avoid adapting the finite difference scheme at each point which interacts with the boundary as above in order to keep the implementation as simple as possible. One possible solution is to fill in the value of the field on the "leg" of the field line, $f_3$, and then use the standard finite difference scheme to compute the parallel derivatives. We call this scheme "Leg Value Fill" (LVF). This involves an extrapolation which needs to be accurate enough to not degrade the accuracy of the FD scheme.

We start from the Taylor expansions, truncated to third order, of $f_1, f_2, f_3$

[1] $f_b$ may also be set from evolved fields inside the computational domain. The problem of how to set $f_b$ from these quantities not unique to FCI, but in general may involve an extrapolation onto the boundary. FCI would then have an additional interpolation along the boundary to the intersection point.

about the boundary:

$$f_1 = f_b - (l_1 + l_2) f'_b + \frac{1}{2}(l_1 + l_2)^2 f''_b - \frac{1}{6}(l_1 + l_2)^3 f'''_b, \tag{5}$$

$$f_2 = f_b - l_2 f'_b + \frac{1}{2} l_2^2 f''_b - \frac{1}{6} l_2^3 f'''_b, \tag{6}$$

$$f_3 = f_b + l_3 f'_b + \frac{1}{2} l_3^2 f''_b + \frac{1}{6} l_3^3 f'''_b. \tag{7}$$

We then use eqs. (6) and (7) to get the first derivative at the boundary

$$f'_b = \frac{1}{l_2 l_3^2 + l_2^2 l_3} [l_2^2 f_3 + (l_3^2 - l_2^2) f_b - l_3^2 f_2] - \frac{1}{6} l_2 l_3 f'''_b. \tag{8}$$

As $f'''_b$ is unknown, this is second-order accurate. Similarly, we can also get the second derivative:

$$f''_b = \frac{2[l_2 f_3 - (l_2 + l_3) f_b + l_3 f_2]}{l_3 l_2^2 + l_2 l_3^2} - \frac{(l_3^2 - l_2^2)}{3(l_2 + l_3)} f'''_b + ... \tag{9}$$

The error in this expression is first order, except for the special case where $l_2 = l_3$ and eq. (9) reduces to the standard central difference scheme.

We can combine eqs. (6) and (7):

$$f_b = \frac{l_2 f_3 + l_3 f_2}{l_2 + l_3} + \frac{f''_b}{2} \frac{l_2 l_3^2 + l_2^2 l_3}{l_2 + l_3} + ... \tag{10}$$

Note that the error term ($f''_b$) is second order in the $l_{1,2,3}$ lengths, so the value at the boundary is determined to second order accuracy. In order to do this, $f_3$ must be set to

$$f_3 = f_b \frac{l_2 + l_3}{l_2} - \frac{l_3}{l_2} f_2 - \frac{f''_b}{2} \frac{l_2 l_3^2 + l_2^2 l_3}{l_2 + l_3} + ... \tag{11}$$

This result can then be used in an arbitrary finite difference scheme to give the parallel derivatives of $f_2$.

For example, putting eq. (11) into the standard $2^{\text{nd}}$-order accurate central difference for the first derivative:

$$f'_2 = \frac{f_b \frac{l_1}{l_2} + f_2(1 - \frac{l_1}{l_2}) - f_1}{2 l_1} - \frac{f''_b}{2} \frac{l_2 l_3^2 + l_2^2 l_3}{l_2 + l_3} + ... \tag{12}$$

It can be seen that this result is still second-order in $l_1, l_2, l_3$.

We can actually go further and get a $3^{\text{rd}}$-order accurate scheme. Insert

eqs. (8) and (9) into eq. (5):

$$\begin{aligned} f_1 = &-f_b \frac{l_1(l_1+l_2+l_3)(l_2+l_3)}{l_2^2 l_3 + l_2 l_3^3} \\ &+ f_2 \frac{l_3}{l_2^2 l_3 + l_2 l_3^2}[(l_1+l_2)l_3 + (l_1+l_2)^2] \\ &+ f_3 \frac{(l_1^2 l_2 + l_1 l_2^2)}{l_2^2 l_3 + l_2 l_3^2} \\ &- f_b''' \frac{1}{6}[l_1^2(l_1+l_2+l_3) + 2l_1 l_2 l_3 + l_2^2 l_3], \end{aligned} \tag{13}$$

drop the $f_b'''$ term and rearrange for $f_3$:

$$\begin{aligned} f_3 = &\frac{l_2^2 l_3 + l_2 l_3^2}{(l_1^2 l_2 + l_1 l_2^2)} f_1 \\ &+ f_b \frac{l_1(l_1+l_2+l_3)(l_2+l_3)}{(l_1^2 l_2 + l_1 l_2^2)} \\ &- f_2 \frac{l_3}{(l_1^2 l_2 + l_1 l_2^2)}[(l_1+l_2)l_3 + (l_1+l_2)^2]. \end{aligned} \tag{14}$$

$f_3$ is now known to third order, and can again be inserted into a standard finite difference scheme.

These two schemes, for second- and third-order, use the points along the field line which are already used in the second-order FCI parallel derivative operator. Higher order schemes can be derived along similar lines, but these require more points along the field lines. These could be generated at the same time as the initial field line tracing.

It is natural to ask if this scheme has consequences for field lines that intersect the boundary at shallow angles, or equivalently with low perpendicular resolution grids. Magnetic field lines might be so shallow as to intersect many perpendicular planes before hitting the boundary. That is, the intersection point on the adjacent plane may be outside the last grid point but still inside the computational domain. We don't anticipate this to be a problem, as for this case, the LVF scheme changes from an extrapolation in the parallel direction, to an interpolation which is often more numerically stable. This can be seen by reducing the tilt of the field line in fig. 2. When the field line is angled such that $f_3$ now lies above the boundary (but still below $f_2$ in the perpendicular direction), then $f_b$ must be now further along the field line from $f_2$ than $f_3$.

We have also derived an expression for $f_2'$ based on a non-uniform grid. Instead of extrapolating to find values on the field line "leg", one can use the value on the boundary directly, but now the finite difference scheme for the parallel derivative must be adapted in order to maintain the second order accuracy. The second order accurate central difference for parallel derivative using this scheme is

$$f_2' = \frac{f_b \frac{l_1}{l_2} + f_2(\frac{l_2}{l_1} - \frac{l_1}{l_2}) - \frac{l_2}{l_1} f_1}{l_1 + l_2}. \tag{15}$$

However, when we tested this approach in a python toy model, we found that this scheme was more prone to numerical instabilities.

Further boundary condition schemes have also been investigated, such as asymmetric or one-sided differences. For these types of schemes, the field line needs to be traced further to the two immediately adjacent poloidal slices. However, the LVF scheme appears to demonstrate the best numerical properties and is the simplest to implement.

#### *2.2.2. Arbitrary geometry*

The boundary scheme presented here is well-suited to a logical rectangular mesh, or the case where limiters are infinitesimally thin and so do not present a face to the magnetic field in the perpendicular direction, or mask the perpendicular grid. While this scheme also works in the case of more complex material boundaries, the problem is a more general one of how to represent the material geometry numerically. The mesh has to either follow the geometry, or grid cells must be "masked" where they intersect the material walls and the equations not evolved there. A masked mesh complicates not just the interpolation for the LVF boundary scheme, but also perpendicular operators and boundary conditions.

Currently, BOUT++ uses a logical rectangular mesh with optional branch cuts to handle X-points. Recent work[18] has enabled this grid to follow the material boundaries more accurately. Future work to upgrade BOUT++ will also explore grids which can handle complex machine geometries, building on the work presented here.

### *2.3. Verification*

An important part of testing a numerical model is verifying that it correctly implements the mathematical model. Validating that the mathematical model correctly represents reality is a separate consideration. Given that it is often the case that an analytical solution cannot be constructed for a mathematical model, it is necessary to use a different technique, such as the Method of Manufactured Solutions[23–25] (MMS). With MMS, an arbitrary "manufactured" solution is imposed, and the mathematical model is applied to this solution. This manufactured solution is in general not an exact solution, however, the "remainder" may be added to the numerical model as source terms such that the manufactured solution now *is* an exact solution of the modified model. The error is defined as the difference between the numerical solution and the manufactured solution. Details on how the MMS framework is implemented in BOUT++ can be found in Ref. [7].

BOUT++, including the FCI method, has been successfully verified using MMS in periodic domains[7]. In this work we use MMS to verify the $2^{\text{nd}}$- and $3^{\text{rd}}$-order LVF boundary condition scheme, as well as to verify different interpolation methods (section 3). The same physics model, computational domain and magnetic field as in Ref. [7] were used, which we briefly restate

here. Two coupled differential equations were evolved for a single time-step:

$$\begin{aligned}\frac{\partial f}{\partial t} &= \nabla_{\parallel} g + D(dy)^2 \nabla_{\parallel}^2 f \\ \frac{\partial g}{\partial t} &= \nabla_{\parallel} f + D(dy)^2 \nabla_{\parallel}^2 g\end{aligned} \tag{16}$$

where $D = 10$ is an artificial diffusivity used purely for numerical stability. A sheared slab with dimensions $L_x = 0.1$m, $L_y = 10$m, $L_z = 1$m in the radial, parallel and binormal directions, respectively, and magnetic field $(B_x, B_y, B_z) = (0, 1, 0.05 + (x - 0.05)/10)$ was used. The manufactured solution used was

$$f = \sin(\bar{y} - \bar{z}) + \cos(t)\sin(\bar{y} - 2\bar{z}), \tag{17}$$

$$g = \cos(\bar{y} - \bar{z}) - \cos(t)\sin(\bar{y} - 2\bar{z}), \tag{18}$$

where $\bar{y}, \bar{z}$ are normalised to be between 0 and $2\pi$. The diffusion terms in eq. (16) scale with $dy^2$ and so do not affect the convergence of the error on $\nabla_{\parallel}$. As in Ref. [7], we scale the grid in $y$ and $z$ simultaneously.

Figures 3 and 4 show the scaling of the MMS errors for $f, g$ for the $2^{\text{nd}}$- and $3^{\text{rd}}$-order LVF schemes, respectively, implemented in BOUT++. The interpolation scheme used was cubic Hermite splines (see section 3). The two schemes produce almost identical results, as the limiting factor on the error scaling is the finite difference scheme, which is second order.

It should be noted that because the $3^{\text{rd}}$-order LVF scheme relies on "upstream" information (i.e. points away from the boundary), it gets stuck in corners, where the field line leaves the boundary in both the forward and backward directions. In these cases, the boundary condition cannot be applied, as is the case for the slab topology presented here. As this is not possible, the results shown here are where the $z$-direction is periodic but the $y$-direction is not. Switching which directions are periodic changes the order by only a fraction of a percent.

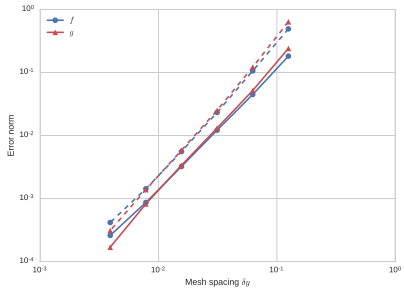



Figure 3: Error scaling for the $2^{\text{nd}}$-order LVF scheme in BOUT++. Solid lines are the $l_2$ norm, dashed lines are the $l_\infty$ norm (i.e. max. error).

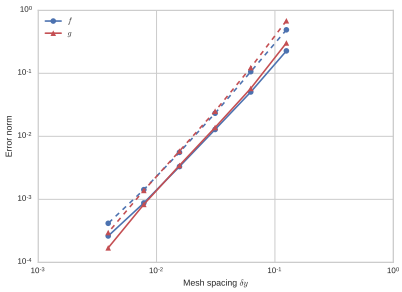



Figure 4: Error scaling for the 3rd-order LVF scheme in BOUT++. Solid lines are the $l_2$ norm, dashed lines are the $l_\infty$ norm (i.e. max. error).

### *2.4. Limiters*

While true arbitrary shaped boundaries have not yet been implemented in BOUT++ due to the reasons stated above, we have made the first steps by implementing an infinitesimally thin poloidal limiter. Field lines either hit the limiter on the front/back face or they miss the limiter altogether and pass behind/in front of it. Thus, no masking of the perpendicular grid is required, which would complicate operators in this plane. The limiter is located halfway between the last and first $y$-planes.

Limiters are implemented in BOUT++ as any function of $(x, z)$ (i.e. on the perpendicular plane) that passes through 0, with positive values indicating the material walls. This enables arbitrarily shaped limiters to be easily created.

The implementation of the limiter is very simple: field lines that end on the $y = 0$ slice can check if they hit the limiter by evaluating the limiter function described above. If the result is positive, the field line is added to the boundary region.

## 3. Interpolation

The FCI method relies on interpolation in order to work, and it is the interpolation which is the most computationally expensive part of the technique (outside of the initial field line tracing, which only needs to be done once for static magnetic fields). It is therefore important to understand how much of an impact the interpolation makes on the accuracy and efficency of the parallel derivate operator. We have implemented three different interpolation methods - bilinear, four-point Lagrange and Hermite splines. The choice of interpolation scheme is made at runtime.

After nearest-neighbour interpolation, bilinear interpolation is one of the most basic forms of interpolation in two dimensions, and consists of two sets of linear interpolation: first in one direction, then in the other.

Lagrange polynomials ensure that the interpolated function goes through the data points exactly. Similarly to the bilinear interpolation, one dimensional

polynomials are used to interpolate in each dimension successively. An $n^{\text{th}}$ order accurate scheme needs to use polynomials of degree at least $n$, which in turns requires at least $n+1$ data points. Higher order polynomials can be used, but these are prone to over-fitting and spurious oscillations between the data-points. A $3^{\text{rd}}$-order (4 point) 2D Lagrange interpolation is implemented in BOUT++.

Lastly, Hermite splines are piecewise polynomials that use the first derivative of the interpolant to act as a tension parameter, ensuring that the interpolated function is $C^1$ continuous. Such splines are computationally more expensive than splines without tension parameters, as the first derivative needs to be evaluated several times for each interpolation. A $3^{\text{rd}}$-order Hermite spline scheme is used in BOUT++ as the default interpolation method for FCI. This is the choice of interpolation scheme used in the original FCI papers[1, 2].

We use the two-field wave model (eq. (16)), and verify the interpolation schemes using MMS (see section 2.3). The results are summarised in fig. 5. All of the interpolation schemes recover the correct $O(dy^2)$ scaling on $\nabla_\parallel$. The Lagrange and Hermite spline interpolation methods achieve identical results, while, on average, the absolute value of the error using the bilinear scheme is more than triple that of the former two schemes. This can be understood from looking at the error scalings on the interpolation schemes. While the Lagrange and Hermite spline methods have different coefficients on their error estimates, in both cases the errors scale like $O(dy^3)$, which is better than the error scaling on the finite difference scheme used here. Bilinear interpolation, on the other hand, has the same order as the finite difference scheme, $O(dy^2)$.

The Hermite spline interpolation is roughly $\sim 45\%$ more computationally expensive than the bilinear interpolation, and $\sim 20\%$ more expensive than the Lagrange polynomials due to the need to evaluate several derivatives. However, it does ensure that the interpolated function is $C^1$ continuous, which may be advantageous, especially for non-linear simulations.

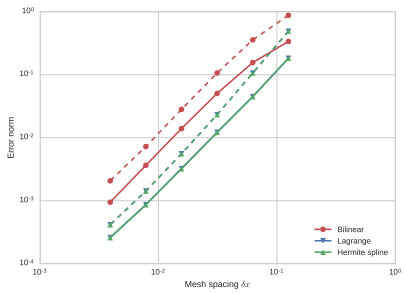



Figure 5: Error scaling of the field $f$ for three different interpolation schemes. Solid lines are the $l_2$ norm, dashed lines are the $l_\infty$ norm

## 4. 3D magnetic fields

The FCI technique has been demonstrated and used in sheared slab[1], cylindrical[1], X-point and island[2, 3], and tokamak[4] magnetic geometries. Here we demonstrate for the first time its use in stellarator-like fields. This magnetic geometry is fully 3D, but has "extrinsic" curvature, i.e. the curvature has to be handled by a bracket operator in the physics model, rather than through the metric tensor. Note that this is a limitation of the current version of BOUT++, and not of the FCI method in general.

### *4.1. Stellarator geometry*

Due to the BOUT++ limitations described above we implement a "straight stellarator", similar to a screw-pinch. The equilibrium is created by specifying coils and compute $\vec{B}$ from Ampère's law. We use four coils, defined by the position $\vec{R}$ of the $k$-th coil which is:

$$\begin{aligned}\vec{R_k}(\varphi) = (x_0 + r_{coil}\cos(\tfrac{1}{2}k\pi + \iota\varphi))\hat{\vec{x}} \\ + (z_0 + r_{coil}\sin(\tfrac{1}{2}k\pi + \iota\varphi))\hat{\vec{z}},\end{aligned} \tag{19}$$

where $(x_0, z_0)$ is the centre of the domain, $r_{coil}$ is the radius of the coil, $\iota$ is the rotational transform of the coils and the current in the $k$-th coil is given by

$$I_k = (-1)^k I_{coil}, \tag{20}$$

with $I_{coil}$ an input parameter.

The magnetic field at a point in space can then be computed as a sum of contributions from the coils:

$$\begin{aligned}B_x(x,y,z) &= \sum_{k=0} I_k \frac{C}{r_k^2}\sin(\theta_k) \\ B_z(x,y,z) &= \sum_{k=0} -I_k \frac{C}{r_k^2}\cos(\theta_k)\end{aligned} \tag{21}$$

where $r_k$ is the distance (in the $(x,z)$ plane) to the $k$-th coil, $\theta_k$ is the azimuthal angle to the coil, $C$ is some nature of constant. We now have expressions for the magnetic field components which can be used as inputs to an FCI grid generator, Zoidberg, (see appendix Appendix A.2) in order to trace the magnetic field and produce the field line maps required for BOUT++.

Figure 6 shows the Poincaré plot at three different $y$ locations, demonstrating the existence of flux surfaces.

We would also like to be able to initialise fields on flux surfaces. While flux surfaces do exist for this magnetic topology, we do not have an expression for $\psi$, the poloidal magnetic flux. Instead, we can use Zoidberg to construct numerical approximations to the flux surfaces. By launching field lines from uniformly spaced radial positions, from the magnetic axis to one edge of the box, and by following them many times around the periodic domain in $y$, flux surfaces are

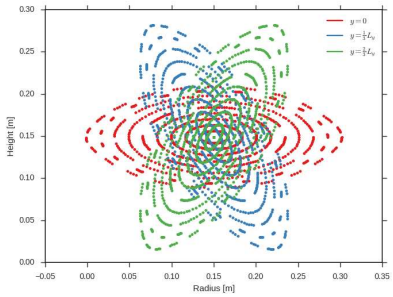



Figure 6: Poincaré plot of a straight stellarator at three $y$ planes

eventually traced out. Values in $[0, 1)$ are then assigned to the field lines according to their initial radial position, and these values then interpolated onto the simulation grid. Points outside the last closed flux surface can be assigned the value 1. The resulting scalar field is numerical approximation to (normalised) $\psi$. Initial conditions for the simulation fields can then be constructed in terms of this approximation to $\psi$ and are therefore flux functions, up to the accuracy of the field line tracing and the interpolation onto the grid. The $\psi$ approximation is used only in the initialisation, and does not appear in the simulations.

## 5. Simulations

### *5.1. Limiter*

We present here preliminary results showing how FCI is able to handle complex 3D magnetic geometry, including first steps towards arbitrary boundaries. The magnetic geometry is a "straight stellarator" as described in section 4.1. The computational domain is a box, periodic in the $y$-direction, with Dirichlet boundary conditions in $(x, z)$.

For these initial simulations, we use a very simple parallel diffusion model:

$$\frac{\partial f}{\partial t} = D_{\parallel} \nabla_{\parallel}^2 f, \tag{22}$$

where $f$ is some scalar field (which we refer to as density), and $D_{\parallel}$ is the parallel diffusivity. Using this model, an initial perturbation will diffuse along the field lines, tracing out flux surfaces. Due to the Dirichlet boundary conditions, density on field lines that hit the boundary will quickly decay away, effectively creating a last closed flux surface (LCFS). Turning on the limiter will therefore change the position of the LCFS.

Figure 7 shows the initial condition:

$$\begin{aligned} f(x, y, z; t = 0) = 100\,&\mathrm{gauss}(x - 0.2, 0.011) \\ &\times \mathrm{gauss}(z - 0.15, 0.3) \sin^6(\tfrac{1}{2} y), \end{aligned} \tag{23}$$

where the normalised Gaussian with width $w$ is given by $\text{gauss}(x, w) = \exp[-x^2/(2w^2)]/(w\sqrt{2\pi})$. The initial condition is a blob, spatially localised off-axis in $(x, z)$, with a wide distribution in $y$.

The thin dot-dashed white lines in fig. 7 show the locations of flux surfaces. In the absence of a limiter, the initial perturbation crosses most of the flux surfaces, whereas with a limiter, it is mostly outside the LCFS. Snapshots of $f$ at late times, with and without a limiter, are shown in figs. 8b and 8d. The density quickly diffuses along the field lines, either hitting the $(x, z)$ edges, or the limiter. In either case, the field is cut off at the respective LCFS.

Figure 8 show the results of two simulations of the diffusion model (eq. (22)) at the same simulation time, figs. 8c and 8d have a circular limiter at $y = 0$ centred on $x = 0.15, z = 0.15$ with radius $r = 0.06$. Figures 8b and 8d show slices of the $(x, y)$ plane half-way through $z$, whereas figs. 8a and 8c are slices of the $(x, z)$ plane at $y = 0$. The vertical solid white lines in fig. 8d and the solid white circle in fig. 8c show the position of the limiter. Note that the limiter is really infinitesimally thin, so presents surfaces only in the $(x, z)$ plane and has no $y$-extent.

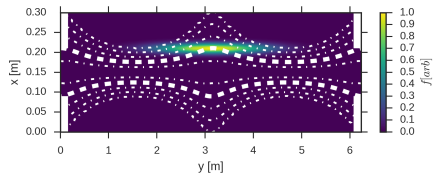



Figure 7: Heat map of initial condition for $f$ in diffusion model in the $(x, y)$ plane at $z = 0.15$. Solid white lines indicate size and position of limiter. Dashed white lines indicate position of last closed flux surface. Dot-dashed lines show positions of flux surfaces.

### *5.2. Upscaling*

As with traditional field-aligned techniques, one of the *raisons d'être* of the FCI technique is the ability to use a low number of points in the parallel direction in order to resolve the relevant physics of a model. Unfortunately, this has a downside when it comes to visualising the data. Typically, visualisation programs use some nature of interpolation in the Cartesian (simulation grid) directions in order to show smoother images. Because the magnetic field is not aligned with the grid, and structures in the data are typically aligned with the magnetic field, this results in rather blocky artefacts. We can reduce or remove these artefacts by first upscaling, i.e. increasing the resolution in the parallel direction, the data ourselves. If we assume the scalar field is slowing varying along the magnetic field line (which is an assumption of FCI itself), we can linearly interpolate along the field line to reconstruct the scalar field at higher parallel resolution.

The upscaling technique we use is as follows. First, as with the usual FCI method, interpolate the data onto the field line end points in one direction.

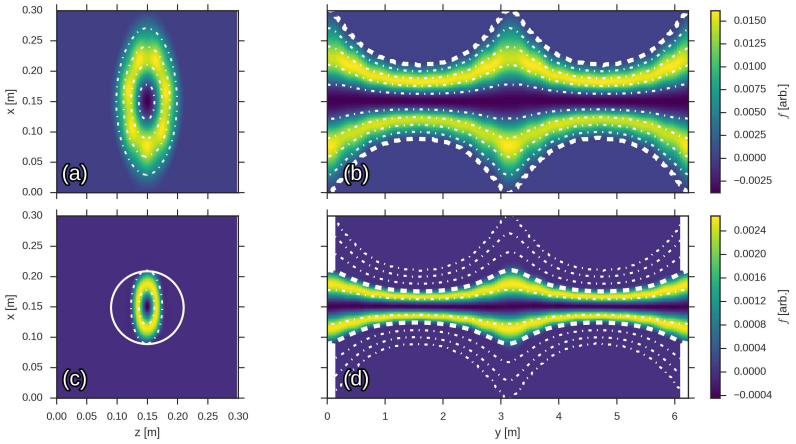



Figure 8: Heat map of $f$ in diffusion model at $t = 400$ . (a): no limiter, (b): circular limiter. Solid white lines indicate size and position of limiter. Dashed white lines indicate position of last closed flux surface. Dot-dashed lines show positions of flux surfaces. Note that these figures have been upscaled in post-processing following the procedure outlined in section 5.2.

Then, use a linear interpolation between the start and end points to get the desired number of additional points. As well as interpolating the data, the $x, z$ displacements should also be linearly interpolated, which saves having to re-integrate the magnetic field. We now have a "cloud" of data on new points. Depending on the visualisation program, these new data can be interpolated themselves back onto a higher resolution rectangular grid, or left as a semi-unstructured grid.

Figure 9 contrasts the result of using this upscaling against the original data. In the original data, there are clear unphysical lobes or fins which are aligned in the $y$ direction, although the simulation is well resolved. In the upscaled version, there are still lobes, but they are now much smaller, and it is now easier to see how the density follows the field lines.

One issue with this upscaling algorithm is that it may give "strange" results when the data are not field-aligned, for example, as with initial conditions or injected sources. In this case, the artefacts are now "blocky" in the parallel direction. Note also that such structures will likely not be well resolved by either FCI or field-aligned approaches.

### *5.3. Numerical diffusion*

There is some perpendicular (cross-field) diffusion from the numerical scheme, even in the model with only parallel derivatives (eq. (22)), due to, e.g. the interpolation scheme. The numerical diffusion in FCI has already been characterised in axisymmetric magnetic geometries[1, 2, 4]. Here we present an estimate of the numerical diffusion for the straight stellarator topology. The expectation is that this should not be substantially different from the previous results[1].

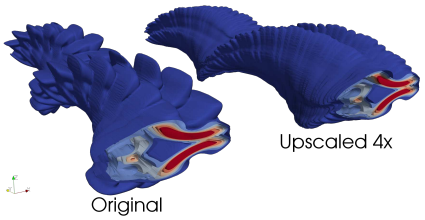



Figure 9: Visualisation of diffusion model (eq. (22)) in ParaView, original data on the left, upscaled by factor 4 on the right. The presence of "fins" can be seen in the original (left) data. These are caused by the visualisation program interpolating in the Cartesian directions, rather than along the magnetic field. In the upscaled (right) version, the data has been interpolated in the parallel direction in order to reduce these fins.

Using the diffusion model (eq. (22)) and initialising $f$ such that $\nabla_\parallel^2 f = 0$, the numerical diffusion can be estimated using

$$\frac{df}{dt} = D_\perp^{\text{eff}} \nabla_\perp^2 f \tag{24}$$

where $D_\perp^{\text{eff}}$ is the effective perpendicular numerical diffusivity. At each time-step, $df/dt$ and $\nabla_\perp^2 f$ can be saved and $D_\perp^{\text{eff}}$ can be computed with

$$D_\perp^{avg} = \langle \|\partial_t f\| / \|\nabla_\perp^2 f\| \rangle, \tag{25}$$

where $\|\|$ is the 2-norm, and angle brackets indicate time average over latter half of simulation. The time average is over the second half of the simulation in order to ignore the effect of initial transients.

In order to measure $D_\perp^{\text{eff}}$ we need to ensure that the parallel derivatives are zero, as this would appear to transport $f$ in the perpendicular plane. To do this, $f$ must be initialised to a flux-function (i.e. constant on flux surfaces). Because we do not have an expression for $\psi$, we must construct a numerical approximation to $\psi$ as described in section 4.1, which can then be used to set an initial condition that is constant on flux surfaces. The initial condition is a Gaussian in $\psi$,

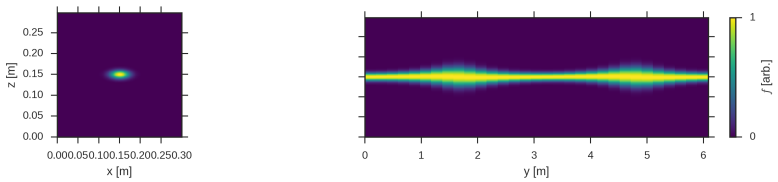
$$f(\psi; t = 0) = A \exp(-(\psi - \psi_0)^2/(2\Delta^2)), \tag{26}$$

with $A = 1$, $\psi_0 = 0$ and $\Delta = 0.1$. Simulations were run up to $100t$, at fixed $n_y = 16$, with $n_x = n_z \in \{16, 32, 64, 128, 256\}$.

The results are summarised in fig. 11. The overall scaling of $D_\perp^{\text{eff}}$ with the perpendicular resolution is of order 2.67, and the absolute values are broadly in line with Ref. [1], despite the magnetic topology there being axisymmetric.

## 6. Conclusions and discussion

We have demonstrated a numerical scheme for parallel boundaries, where magnetic field lines intersect the material wall, for use with the Flux-Coordinate




(a) (b)

Figure 10: Initial condition for the numerical diffusion test case. (a): $(x, z)$ plane at $y = 0$, (b): $(y, z)$ plane at $x = 0.15$.

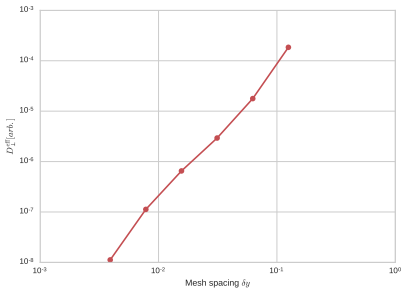



Figure 11: Numerical effective perpendicular diffusion as a function of mesh spacing.

Independent (FCI) method for numerical derivatives parallel to the magnetic field. The scheme for Dirichlet boundary conditions is based on a Taylor expansion about the boundary in order to extrapolate the field onto the "leg" of the field line outside the boundary. Second- and third-order accurate versions of the scheme have been derived. In the case of shallow grazing angles, where the field line intersects the next poloidal plane before the material wall, this scheme corresponds to an interpolation in the parallel direction, and so arbitrary-shaped material walls may be handled easily with the same scheme. The Method of Manufactured Solutions (MMS) has been used to rigorously verify the accuracy and correct implementation of the boundary scheme.

The feasibility of performing simulations in non-axisymmetric magnetic configurations using the FCI method has been demonstrated, with a simple diffusion model in a straight, stellarator-like magnetic field. An initial Gaussian blob in a simple diffusion model traces out flux surfaces. The inclusion of a poloidal limiter reduces the radial extent of the flux surfaces thereby traced out. Non-axisymmetry has been shown to not substantially affect the effective numerical diffusivity.

A novel technique for reducing blocky artefacts in visualisations during post-processing has also been demonstrated. By linearly interpolating both the data to be visualised and the field line displacement map at the same time, the parallel resolution of the data can be up-sampled, and the new data re-interpolated onto a higher resolution grid. Smoother, contours can then be produced, with fewer

artefacts not present in the data.

An open question remains on the computational efficency of FCI. Obviously, this does depend on the exact interpolation method used, the perpendicular grid resolution, the finite difference scheme, the degree of anisotropy in the physics, etc., but what is not obvious is when the cost of the FCI overheads is outweighed by the advantage in the parallel resolution.

An important consideration is that FCI is designed for complex magnetic topologies which are difficult to represent or capture with conventional field-aligned grids. For example, the island divertors in a stellarator[26] involve multiple null points as well as large regions of stochastic magnetic field. These would be very challenging to simulate using the usual mesh in BOUT++. Another example would be the snowflake divertor concept[27], which has multiple legs. This has been previously attempted in BOUT++[28], but this study was only able to capture the expanded flux surfaces in the region of the null point, and not the additional legs which are a feature of a second-order null point. Here, then, it is clear that using the FCI method lets us get much further towards simulating plasma in these complex geometries, regardless of the computational cost.

In other situations, it is not so clear-cut that FCI presents a major advantage over a field-aligned grid. Take, for instance, an island perturbation on a tokamak equilibrium. This can be represented in a field-aligned grid simulation by splitting the magnetic field into equilibrium and perturbation caused by the island. A bracket operator can then be used to capture the physics due to the island field. The same method can be used for electromagnetic simulations where the perturbed magnetic field is a function of time. In this case, further study is needed to determine the parameter regime where it is clearly advantageous to use FCI over a field-aligned grid.

*6.1. Acknowledgements*

The authors would like to thank David Dickinson from the University of York for useful discussions and comments.

This work has been carried out with the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014–2018 under grant agreement No. 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

## Appendix A. BOUT++ implementation

*Appendix A.1. Coordinate systems*

While it is technically possible to switch between using the standard BOUT++ mesh and FCI for a given problem, currently there are some technical hurdles. The assumptions on the nature of $x$, $y$, and $z$ in BOUT++ simultaneously limit FCI in the choice of perpendicular coordinates, while lifting some restrictions in the parallel direction. One such assumption is that $y$ identifies the "parallel" direction, which is important for some internal details, e.g. communication. This

creates a conflict between the BOUT++ and FCI meshes, as the BOUT++ coordinate system identifies $\theta$ as the parallel coordinate whereas FCI assumes $\zeta$ is the parallel direction. As such, switching between the two meshes may not be trivial for a given physics model.

Another technical issue is that the current implementation of BOUT++ assumes $z$ is axisymmetric, but makes no such assumption on $y$. Thus, using FCI, it is possible to simulate non-axisymmetric configurations, such as stellarators, which are not possible otherwise, at the cost of complicating the inclusion of curvature effects. Note that these obstacles are not inherent to FCI – merely the particular implementation of FCI in BOUT++. Overcoming these technical limitations is the focus of future work.

*Appendix A.2. Grid generation*

The derivation of the FCI technique is discussed in Refs. [1, 2]; here we discuss its particular implementation in BOUT++. There are three major steps required for FCI: first, the magnetic field lines must be followed from each grid point in both directions, and the intersection points with the adjacent perpendicular planes recorded; secondly, the scalar field must be interpolated at the intersection points; lastly, a finite difference can be applied using the interpolated values.

Following the magnetic field lines, or *field line tracing*, generates a *field line map* that maps a given grid point to its intersection point on the next/previous perpendicular plane. Two field line maps are needed, one for the forwards (positive $y$) and one for the backwards (negative $y$) directions. We construct these field line maps with a tool called Zoidberg, written in python. Zoidberg uses *odeint* from *SciPy*[19] to trace the field lines. The magnetic field can be supplied to Zoidberg either as an tuple of three analytic functions (for $B_x(x, y, z), B_y(x, y, z), B_z(x, y, z)$), or a tuple of arrays which are to be interpolated by *odeint*. The latter form allows general numeric equilibria (from e.g. *VMEC*[20] or *EFIT*[21] files) to be used as input for FCI grids in BOUT++. Field line maps for equilibria specified via analytic functions are very quick to create with Zoidberg, taking approximately 40 seconds on a single thread for a 256x16x256 grid. Numerical equilibria take longer as the interpolation function, in general, is more expensive to compute. The output from Zoidberg is a file containing the field line maps. This is an input to BOUT++ – currently, only time-independent magnetic fields are supported.

Time-dependent magnetic fields could be supported through one of two methods. Firstly, by assuming that the magnetic field can be split into an equilibrium time-independent part, and a time-dependent perturbed part, the interaction of perturbed part with the plasma may be handled via a bracket method. This method can already be employed and needs no modification of BOUT++. The second method would be to evolve the magnetic field, then pause the simulation while Zoidberg is used to update the field line maps. Some modifications of BOUT++ would be needed in order to call an external application and also to reload the grid file. Due to this overhead, one may

choose to evolve the magnetic field on a slower time-scale than the rest of the simulation.

The second step of the FCI method, interpolation, is handled internally in BOUT++. At each time-step, all fields which are to be acted upon by parallel derivative operators must be interpolated at the points held in the field line maps. For details of the specific interpolation techniques used in BOUT++, see section 3.

The finite difference schemes and their implementation in BOUT++ are discussed in [7].

*Appendix A.3. Boundaries*

The boundary conditions in BOUT++ are set at run-time, including the choice of making the $y$ and/or $z$ boundaries periodic for FCI. Currently, non-periodic $z$ boundaries are only supported by the FCI parallel derivative operators in BOUT++, and not by any other spatial operator. Future work will address supporting non-periodic $z$ boundaries generically.

During the initialisation stage in BOUT++, the field line maps are read in, the field lines that hit the edge are detected, and for each such field line a data structure of information required for the boundary condition is appended to a vector. A separate vector of these structures is kept for the forward and backward directions, and each vector is stored in a *BoundaryRegionPar* class. When the boundary conditions are applied to a field during the course of the simulation, these vectors can be iterated over, and the LVF scheme is applied to populate the relevant points.

The *BoundaryRegionPar* class needs to know some pieces of information about the field lines that intersect the boundaries. These are the originating index point, the index-space coordinates of the intersection with the boundary, and the angle and distance to the boundary. Briefly, the algorithm to collect this information is implemented as follows: first determine which, if any, edges the field lines intersect; then find the coordinates of the intersection point. For simple, planar boundaries, determining the intersection point is a trivial application of trigonometry; for more complex boundaries, determining where the field lines intersect the material walls may need to be done with the field line tracing procedure. In either case, once the intersection point with the boundary is determined, the distance along the field line, and the angle the field line makes to the boundary can be computed. While the angle of intersection is not used in the present work, it may be useful in more sophisticated boundary conditions, e.g. Loizu[22] boundary conditions for plasma pre-sheaths in the divertor region of tokamaks, where the boundary ion velocity is proportional to the sine of the angle of intersection.

The value of the field on the boundary can be set to either a constant value, or an analytic function of space and/or time. More sophisticated schemes may require the boundary value to depend on interior values, possibly of multiple fields (e.g. the Loizu schemes mentioned above). How to set the boundary value in such cases is a problem not unique to FCI. In general, one has to extrapolate

onto the boundary; with FCI, an extra interpolation along the boundary to the intersection point is also required.


[1] F. Hariri and M. Ottaviani, Computer Physics Communications **184**, 2419 (2013).

[2] F. Hariri, P. Hill, M. Ottaviani, and Y. Sarazin, Physics of Plasmas **21**, 082509 (2014).

[3] P. Hill, F. Hariri, and M. Ottaviani, Physics of Plasmas **22**, 042308 (2015).

[4] A. Stegmeir, D. Coster, O. Maj, K. Hallatschek, and K. Lackner, Computer Physics Communications **198**, 139 (2016).

[5] B. Dudson, M. Umansky, X. Xu, P. Snyder, and H. Wilson, Computer Physics Communications **180**, 1467 (2009).

[6] B. D. Dudson, a. Allen, G. Breyiannis, E. Brugger, J. Buchanan, L. Easy, S. Farley, I. Joseph, M. Kim, a. D. McGann, J. T. Omotani, M. V. Umansky, N. R. Walkden, T. Xia, and X. Q. Xu, Journal of Plasma Physics **81** (2014), 10.1017/S0022377814000816, arXiv:1405.7905 .

[7] B. Dudson, J. Madsen, J. Omotani, P. Hill, L. Easy, and M. Løiten, Physics of Plasmas **23**, 062303 (2016), arXiv:1602.06747 .

[8] P. W. Xi, X. Q. Xu, X. G. Wang, and T. Y. Xia, Physics of Plasmas **19** (2012), 10.1063/1.4751256.

[9] B. D. Dudson, X. Q. Xu, M. V. Umansky, H. R. Wilson, and P. B. Snyder, Plasma Physics and Controlled Fusion **53**, 054005 (2011).

[10] P. Snyder, R. Groebner, J. Hughes, T. Osborne, M. Beurskens, a.W. Leonard, H. Wilson, and X. Xu, Nuclear Fusion **51**, 103016 (2011).

[11] N. R. Walkden, B. D. Dudson, and G. Fishpool, Plasma Physics and Controlled Fusion **55**, 105005 (2013), arXiv:arXiv:1307.5234v1 .

[12] J. R. Angus, M. V. Umansky, and S. I. Krasheninnikov, Physical Review Letters **108**, 1 (2012).

[13] B. Friedman, T. A. Carter, M. V. Umansky, D. Schaffner, and B. Dudson, Physics of Plasmas **19** (2012), 10.1063/1.4759010, arXiv:1205.2337 .

[14] B. W. Shanahan and B. D. Dudson, Journal of Physics: Conference Series **561**, 012015 (2014).

[15] X. Q. Xu, M. V. Umansky, B. Dudson, and P. B. Snyder, Communications in Computational Physics **4**, 949 (2008).

[16] B. Scott, Physics of Plasmas **8**, 447 (2001).

[17] A. M. Dimits, Physical Review E **48**, 4070 (1993).

[18] J. Leddy, B. Dudson, M. Romanelli, B. Shanahan, and N. Walkden, (2016), arXiv:arXiv:1604.05876v1 .

[19] E. Jones, T. Oliphant, P. Peterson, and Others, "SciPy: Open Source Scientific Tools for Python," (2001).

[20] S. P. Hirshman and J. C. Whitson, Physics of Fluids **26**, 3553 (1983).

[21] L. Lao, H. St. John, R. Stambaugh, A. Kellman, and W. Pfeiffer, Nuclear Fusion **25**, 1611 (1985).

[22] J. Loizu, P. Ricci, F. D. Halpern, and S. Jolliet, Physics of Plasmas **19** (2012), 10.1063/1.4771573.

[23] W. L. Oberkampf and C. J. Roy, *Verification and Validation in Scientific Computing* (Cambridge University Press, New York, NY, 2010).

[24] K. Salari and P. Knupp, "Code Verification by the Method of Manufactured Solutions," Tech. Rep. (Sandia National Laboratories, 2000).

[25] P. J. Roache, *Verification and Validation in Computational Science and Engineering* (Hermosa Publishers, Albuquerque NM, 1998).

[26] P. Grigull, K. McCormick, J. Baldzuhn, R. Burhenn, R. Brakel, H. Ehmler, Y. Feng, F. Gadelmeier, L. Giannone, D. Hartmann, D. Hildebrandt, M. Hirsch, R. Jaenicke, J. Kisslinger, J. Knauer, R. König, G. Kühner, H. Laqua, D. Naujoks, H. Niedermeye, N. Ramasubramanian, N. Rust, F. Sardei, F. Wagner, A. Weller, U. Wenzel, and W7-AS Team, Plasma Physics and Controlled Fusion **43**, A175 (2001).

[27] D. D. Ryutov, Physics of Plasmas **14** (2007), 10.1063/1.2738399.

[28] J. Ma, X. Xu, and B. Dudson, Nuclear Fusion **54**, 033011 (2014).