Computer Physics Communications 318 (2026) 109893

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Computer Programs in Physics

STORM: Scrape-off layer turbulence in tokamak fusion reactors

J.T. Omotani [a,*], D. Dickinson [b], B.D. Dudson [c,b], L. Easy [a,b], D. Hoare [a,d], P. Hill [b], T. Nicholas [a,b], J. Parker [a], F. Riva [a,1], N.R. Walkden [a], Q. Xia [a], F. Militello [a]

[a] United Kingdom Atomic Energy Authority, Culham Campus, Abingdon, Oxfordshire, OX14 3DB, UK
[b] York Plasma Institute, Department of Physics, University of York, Heslington, York, YO10 5DD, UK
[c] Lawrence Livermore National Laboratory, Livermore, CA 94550, USA
[d] Department of Physics, University of Bath, Bath, BA2 7AY, UK

ARTICLE INFO

The review of this paper was arranged by Prof. David W. Walker

Keywords:
Plasma
Tokamak
Scrape-off layer
Turbulence
BOUT++

ABSTRACT

The scrape-off layer of a tokamak fusion reactor carries the plasma exhaust from the hot core plasma to the material surfaces of the reactor vessel. The heat loads imposed by the exhaust are a critical limit on the performance of fusion power plants. Turbulent transport of the plasma regulates the width of the scrape-off layer plasma and must be modelled to understand the intensity of these heat loads.

STORM is a plasma turbulence code capable of simulating three dimensional turbulence across the full scrape-off layer of a tokamak fusion reactor, using a drift reduced, collisional fluid model. STORM uses mostly finite difference schemes, with a staggered grid in the direction parallel to the magnetic field. We describe the model, geometry and initialisation options used by STORM, as well as the numerical methods, which are implemented using the BOUT++ plasma simulation framework.

BOUT++ has been enhanced alongside the development of STORM, providing better support for staggered grid methods. We summarise these enhancements, including a detailed explanation of the parallel derivative methods, which underwent a major update for version 4 of BOUT++.

Program summary

Program Title: STORM
CPC Library link to program files: https://doi.org/10.17632/zm3tdfhp9r.1
Developer's repository link: https://github.com/boutproject/STORM
Licensing provisions: GPLv3
Programming language: C++
Supplementary material: Configuration and input files and post-processing scripts to run the example code given in Listings 1, 2, and 3.
Nature of problem: The scrape-off layer region of tokamak fusion reactors carries the plasma exhaust which escapes from the core, confined plasma and reaches material surfaces along open magnetic field lines. The power and particle loads on the material surfaces are a critical limiting factor for the performance of fusion reactors, but are challenging to simulate due to the large fluctuation amplitudes, complex magnetic geometry, and widely separated time- and length-scales. Three dimensional simulations of plasma turbulence are needed to understand the particle and energy transport in the scrape-off layer and provide predictive capability for the design of future reactors.
Solution method: STORM solves a drift reduced, collisional, fluid model for the scrape-off layer plasma. The model is discretised in space using mostly finite difference methods, combined in some places with Fourier methods that take advantage of the toroidal symmetry of the tokamak geometry. The fastest dynamics occur in the direction parallel to the magnetic field, for which a staggered grid is used to avoid the chequerboard instability associated with advective equations [1, sections 6.2, 6.3]. The time solver is a fully implicit, matrix free, variable-step, variable-order method provided by the SUNDIALS library [2]. STORM is implemented using the BOUT++ framework for plasma simulations.

* Corresponding author.
E-mail address: john.omotani@ukaea.uk (J.T. Omotani).
[1] Current address: IRSOL, Via Patocchi, 57 6605 Locarno Monti, Switzerland.

https://doi.org/10.1016/j.cpc.2025.109893
Received 16 December 2024; Received in revised form 3 September 2025; Accepted 2 October 2025

Available online 9 October 2025

References

[1] S. Patankar, Numerical Heat Transfer and Fluid Flow, Hemisphere Publishing Corporation, 1980.
[2] A. C. Hindmarsh, P. N. Brown, K. E. Grant, et al., ACM Trans. Math. Softw. 31 (3) (2005) 363–396.

1. Introduction

Transport of heat and particles through the region of open magnetic field lines in the boundary of a tokamak fusion reactor regulates the loads reaching material surfaces, which limit reactor performance [1]. This region, the scrape-off layer (SOL), is difficult to model as it combines strong plasma-neutral interactions, complex geometry, boundary conditions at material surfaces and large amplitude plasma fluctuations.

The state of the art in first principles modelling of the saturated, statistical steady state of SOL plasma turbulence is represented by codes which implement plasma equations based on collisional fluid closures [2], restricted to low frequency dynamics using drift ordering [3,4]. While the assumptions necessary to construct fluid closures restrict the applicability of these codes, for example they are not suitable for modelling the high performance H-mode [5] where collisionality is low, they are sufficiently tractable computationally for simulation of the entire SOL of present day tokamaks using realistic parameters [6,7]. STORM ('Scrape-off layer Turbulence ORiented Model') is one of several such codes that have been developed by the community, others are FELTOR [8], GBS [9], GDB [10], GRILLIX [11], Hermes [12,13], and SOLEDGE3X [14]. There is also ongoing progress in gyrokinetic modelling of SOL turbulence [15–18].

As transport in a magnetised plasma is much more rapid in the direction parallel to the magnetic field than in the perpendicular directions, turbulent structures tend to be aligned to the magnetic field, having much smaller parallel gradients than perpendicular gradients. Parallel derivative operators are often treated specially to take advantage of this property by allowing one dimension to have its grid spacing increased by two or three orders of magnitude, giving a correspondingly less restrictive Courant–Friedrichs–Lewy (CFL) condition [19,20] and reduced memory requirement. STORM and Hermes are built using the BOUT++ framework [21–23] and are distinguished from the other SOL turbulence codes by calculating derivatives parallel to the magnetic field using a field aligned computational grid. In contrast FELTOR and GRILLIX use the 'Flux Coordinate Independent' (FCI) approach [24] (which has also been implemented in BOUT++ [25,26]) where parallel derivatives are calculated in a locally field aligned way but without a globally field aligned grid; GDB is restricted to limiter configurations without X-points in the poloidal magnetic field, and calculates parallel derivatives in a similar way to what BOUT++ calls the 'shifted metric' scheme, described in section 2.3; GBS and SOLEDGE3X use non-aligned derivatives, GBS on a simple, cylindrical grid and SOLEDGE3X on a flux surface aligned grid. While STORM and Hermes are both built on BOUT++, STORM uses a staggered grid in the parallel direction and finite difference operators, while Hermes uses an unstaggered grid and finite volume operators.

Coherent, field aligned structures known as 'filaments' (or 'blobs' from their cross section perpendicular to the magnetic field) [27] are typically observed in the SOL of tokamaks [28–31], and also stellarators [32]. STORM was originally developed to study the dynamics of filaments using an electrostatic, cold ion, isothermal electron model, in a simplified curved slab geometry, where the filament was imposed as an initial condition [33,34], building on earlier filament studies with BOUT++ [35,36]. After electron temperature evolution was introduced [37], STORM was part of a multi-code validation comparing to filament measurements from MAST [38] and the effects of interactions with other filaments [39] and with background neutral gas [40,41] were studied. Electromagnetic effects were introduced to the equations, and investigated with further filament simulations [42]. A variant, STORM2D, which simulates a plane perpendicular to the magnetic field,

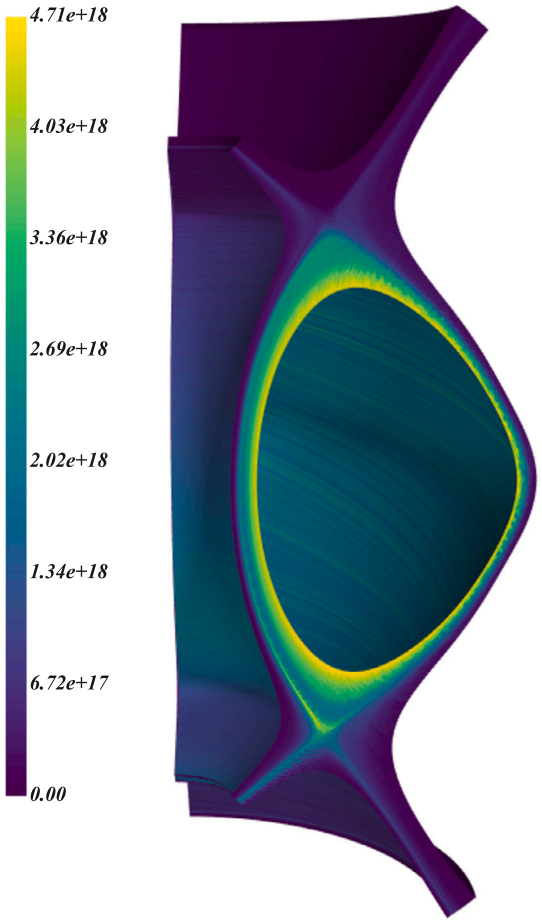


Fig. 1. Density in units of m^{-3}, from a STORM turbulence simulation of MAST pulse #21712. To smoothly visualise the field-aligned turbulent structures, the poloidal resolution has been increased by a factor of 16 using interpolation parallel to the magnetic field.

using closures for the parallel dynamics (which is common in the literature, see for example the review [43]), has been used for studies of both isolated filaments [33,39,44] and turbulence [45–47]. We focus here on the three dimensional version and so do not discuss STORM2D further, for more details see [48,49].

Starting with [6,50], STORM has transitioned to focus on studies of turbulence. Well diagnosed MAST discharges in a suitable regime for modelling by STORM are available. STORM results, illustrated in Fig. 1, were compared to pulse #21712 [6], which is a double-null, ohmic L-mode discharge, providing the first detailed experimental comparison of a three dimensional SOL turbulence simulation to a double-null tokamak experiment. While the fluctuation amplitudes and SOL width were somewhat underestimated, many of the experimentally measured statistical properties of the turbulence were well reproduced by the simulation. Synthetic fast camera signals were constructed from the simulation data, and compared to a database of experimental results that exploit the wide field of view of the fast camera in MAST to enable a detailed characterisation of turbulence localised to the divertor volume [51]. The simulation can reproduce a range of features observed across a wide range of conditions in the experimental database. In addition, by disabling several types of turbulence drive in simulations their

relative effect and importance could be observed. For example it was possible to identify the contrasting role of magnetic curvature in driving flux into the PFR in the inner divertor leg where curvature is aligned with the average pressure gradient, while suppressing flux into the PFR in the outer divertor leg where curvature is in the opposite direction to the average pressure gradient. Other studies have looked at aspects of SOL turbulence using a curved slab geometry [45,50,52].

This paper describes the capabilities of STORM and gives a fuller description of the numerical methods used than is possible in papers focused on simulation results, including upgrades that were made to BOUT++ in support of STORM development. We begin section 2 with a brief overview of the BOUT++ framework to set the scene for the rest of the paper, then in section 2.2 we outline upgrades to the support for staggered grids in BOUT++ so that they are correct in non-slab geometry, and in section 2.3 give an explanation of the various methods used to treat parallel derivatives using field aligned numerical schemes in BOUT++ version 4 on flux surface aligned grids (i.e. excluding the FCI scheme). Section 3 describes STORM, beginning with the model equations and boundary conditions in section 3.1. The numerical methods used in STORM are detailed in section 3.2, with methods for initialising simulations in section 3.3 and the source terms used to sustain background profiles and drive turbulence in section 3.4. The implementation of synthetic Langmuir probe diagnostics is discussed in section 3.5. We summarise in section 4. Appendix A briefly introduces an alternative syntax provided by BOUT++ to allow more efficient thread-based parallelism, Appendix B presents the normalised model equations as implemented in STORM, Appendix C is a brief note on simplification of the curvature operator in a 'slab' geometry, the parallel scaling performance of STORM is discussed in Appendix D, the post processing libraries xBOUT and xSTORM are briefly introduced in Appendix E, and some features for provenance tracking of simulation results are described in Appendix F.

2. BOUT++ updates supporting STORM

BOUT++ is a framework for writing physics codes that provides a general purpose, three dimensional partial differential equation solver in curvilinear coordinates, with specialised features targeted at representing magnetised plasmas in toroidal geometry, especially using drift reduced fluid models. The productivity of physicists developing codes is enhanced by an operator syntax that allows time evolution equations to be expressed in a very similar form to their mathematical expressions, while the underlying library is well optimised to provide efficient performance up to several thousand cores on high performance computing (HPC) clusters. Next we give a very brief overview of BOUT++ to provide context for the rest of the paper; for more detail see [21–23] and the online documentation [53]. In the rest of this section we detail two upgrades to the BOUT++ framework, to staggered grids in section 2.2 and new options for field aligned derivatives in section 2.3, which were critical to the development of STORM.

The primary use case for BOUT++, on which we focus in this paper, is in simulating the edge and SOL region of tokamak fusion reactors. Tokamaks are toroidally symmetric to a good approximation, and so their geometry can be described fully by a two dimensional 'poloidal' cross section. The option to use fully three dimensional geometry to better support cases with no symmetry direction, such as stellarators, has been added to the version 5.0.0 major release of BOUT++ [54], but will not be discussed further here. This paper focuses on the version 4 series of BOUT++ releases which began with version 4.0.0 in 2017 [55], but the methods described are still used in version 5. A grid file provides geometrical information such as the magnetic field strength, metric coefficients for a locally field aligned geometry (see section 2.1), etc. on a grid such as that shown in Fig. 2. The example shown in Fig. 2 is a connected double-null configuration; BOUT++ also supports single-null and disconnected double-null topologies. The non-trivial topology of the grid is handled by representing physical variables on a globally rectangular, block structured mesh, as shown in Fig. 3, which has branch cuts to match the connectivity of the physical mesh. The mesh can be divided into: a 'core', closed flux surface region; one or two SOL regions of open flux surfaces that share a radial edge with the core region; and private flux regions (PFRs), open flux surface regions that share radial edges only with SOL regions. The choice of a globally rectangular mesh in the poloidal plane restricts the number of radial grid points in the inner SOL to be equal that in the outer SOL, and the number in the PFR to be equal to the number in the core for the PFR adjacent to the primary X-point, or number in the core plus the inter-separatrix region for the PFR adjacent to the secondary X-point in a disconnected double-null configuration. More general poloidal-plane grids, which relax this restriction and could support more topologies, for example snowflake or X-point target, might be implemented in future work. The third, toroidal, direction of the grid is taken to be periodic. BOUT++'s naming convention is that the radial direction, which uses the poloidal magnetic flux function ψ as its coordinate, is x; the poloidal direction, whose coordinate will be aligned to the magnetic field $\boldsymbol{B}$, is y; and the (shifted) toroidal direction is z. Simpler geometries, such as a Cartesian slab, can also be defined and are often useful for testing. The coordinates and geometry are discussed further in section 2.1.

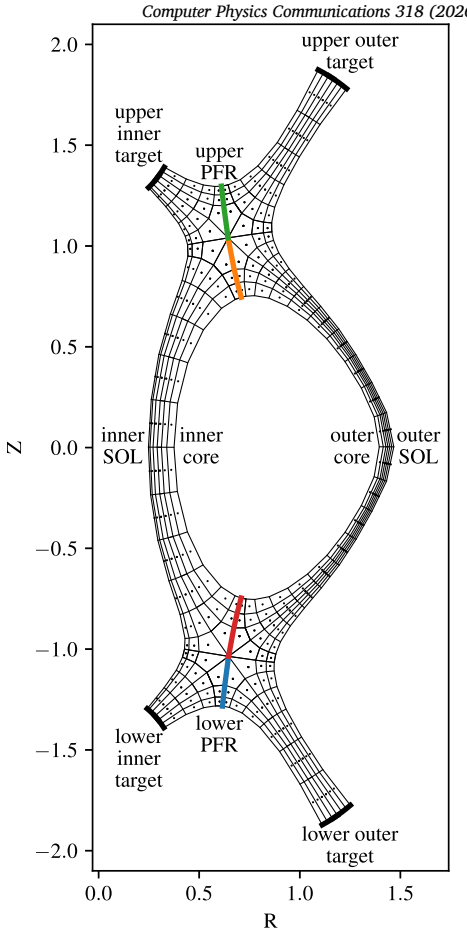


Fig. 2. Example BOUT++ grid for double-null configuration. For clarity, this example grid is much coarser than those usually used for simulations. Targets where the magnetic field intersects material surfaces are shown with thick black lines. The branch cut locations shown in Fig. 3 are highlighted with thick coloured lines. Cell centre grid points are black dots, while cell faces are thin black lines.

The main parallelisation strategy for BOUT++ is domain decomposition using MPI. The two dimensional grid is split between MPI ranks, and the array representing each variable includes a halo of 'guard cells' whose values are set by communicating with neighbouring processes. The branch cuts are required to be on processor boundaries so that their

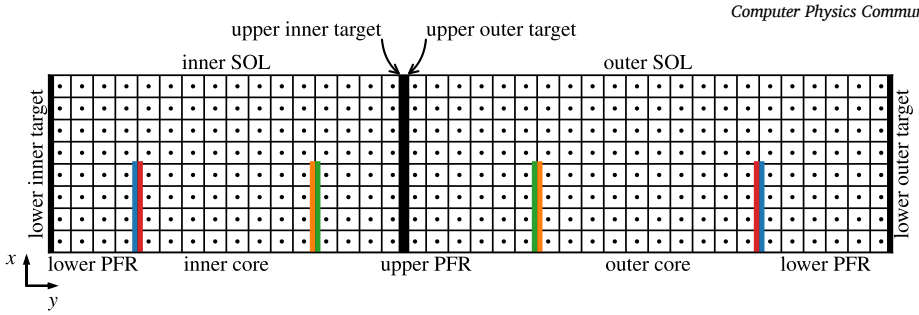


Fig. 3. Logical grid with same structure as Fig. 2. The x-direction is radial and the y-direction is poloidal. For clarity fewer cells in the y-direction are shown here than in Fig. 2. As in Fig. 2 the targets are shown with thick black lines. Branch cuts are shown with thick coloured lines; grid cells adjacent to edges of the same colour communicate with each other, as they are physically adjacent with both coloured edges being located at the single line with the same colour in Fig. 2. Cell centre grid points are black dots, while cell faces are thin black lines.

```
#include <bout/physicsmodel.hxx>
class SimpleWave : public PhysicsModel {
  int init(bool UNUSED(restarting)) {

    solver->add(f, "f");
    solver->add(g, "g");

    return 0;
  }

  int rhs(BoutReal UNUSED(t)) {
    mesh->communicate(f, g);

    ddt(f) = Grad_par(g);
    ddt(g) = Grad_par(f);

    return 0;
  }

  Field3D f;
  Field3D g;
};

BOUTMAIN(SimpleWave);
```

Listing 1: BOUT++ program implementing wave equations.

handling can be delegated to the communication routines. The toroidal direction is not decomposed, in order to allow efficient fast Fourier transforms (FFT) in this periodic dimension. BOUT++ does implement thread based parallelism using OpenMP, in part to compensate for the lack of domain decomposition in the toroidal direction. For technical reasons described in Appendix A, efficient use of thread based parallelism requires a less convenient syntax for the physics code. STORM uses the usual, compact BOUT++ syntax and therefore does not routinely use thread based parallelism.

Codes that are built with BOUT++ define a subclass of the `PhysicsModel` abstract base class. For example, a simple set of wave equations

$$\frac{\partial f}{\partial t} = \nabla_\parallel g \tag{1}$$

$$\frac{\partial g}{\partial t} = \nabla_\parallel f, \tag{2}$$

where the parallel gradient is $\nabla_\parallel = \boldsymbol{b}\cdot\nabla$ and $\boldsymbol{b} = \boldsymbol{B}/B$ is the unit vector in the direction of the magnetic field, can be implemented with the code[2] in Listing 1, which defines a `SimpleWave` subclass of `PhysicsModel`.

[2] Configuration and input files to compile this code and those in the subsequent listings, linked to BOUT++, and run them on a one dimensional, periodic grid are provided in the supplementary material.

The scalar variables f and g are represented by `Field3D` objects `f` and `g`, which are included as member variables of `SimpleWave`. `PhysicsModel` subclasses are required to implement two methods: `init()`, which performs any initialisation needed before the time loop starts; and `rhs()`, which evaluates the time derivatives of the evolving variables. Here `init()` declares to the time solver object `solver` that `f` and `g` should be advanced in time. `rhs()` first calls `mesh->communicate(f, g)` to fill the guard cells of `f` and `g` on each process by communicating with its neighbours, as is necessary before calculating derivatives. Then the evolution equations (1) and (2) are transcribed into C++ code using operators provided by BOUT++: `ddt()` is a convenience macro that fetches the `Field3D` member variable where the time derivative is stored; `Grad_par()` is a finite difference discretisation of the $\nabla_\parallel$ operator. The final part of the code is the macro `BOUTMAIN()`, which provides a standard `main()` function that handles library initialisation and finalisation, and runs the simulation.

Different types of variable are represented by different subclasses of the `Field` abstract base class. `Field3D`, as seen above, is used to represent evolving plasma variables. `Field2D` represents toroidally-symmetric quantities, such as metric coefficients or magnetic field strength. `FieldPerp` represents a slice of a `Field3D` at constant y, which can be useful for certain operations such as inversion of the $\nabla_\perp^2$ operator or applying boundary conditions at the y-boundaries.

2.1. Coordinates and geometry

The standard, toroidally symmetric, tokamak geometry is defined in BOUT++ using locally field aligned flux coordinates [53], which we outline here for completeness. The major radius R, toroidal angle ζ (increasing anticlockwise as seen from above) and vertical position Z define a right handed, cylindrical coordinate system $\{R,\zeta,Z\}$ whose axis is the symmetry axis of the tokamak. A right handed flux coordinate system is given by $\{\psi,\theta,\zeta\}$ where ψ is the poloidal magnetic flux divided by 2π (which we assume here increases from the magnetic axis to the separatrix), θ is some angle-like coordinate which parameterises the poloidal position on each flux surface (increasing in the clockwise direction on an R-Z plane), and ζ is the toroidal angle (the same as for the cylindrical coordinate system). The exact definition of θ is arbitrary, and is decided by the grid generator. A field aligned coordinate system [21,56,57] is defined by

$$x = \psi - \psi_0 \tag{3}$$

$$y = \theta \tag{4}$$

$$z = \zeta - \int_{\theta_0}^{\theta} \frac{\boldsymbol{B}\cdot\nabla\zeta}{\boldsymbol{B}\cdot\nabla\theta} d\theta. \tag{5}$$

This choice ensures, although we do not prove it here, that x and z are constant along magnetic field lines, so that y is a 'field aligned coordi-

nate'. ψ_0 is an arbitrary constant, and θ_0 defines the poloidal position where the x-z grid is orthogonal. Finally we define a set of locally field aligned coordinate systems where for each x-z grid plane at a given θ_l, the local coordinate system is

$$x = \psi - \psi_0 \tag{6}$$

$$y = \theta \tag{7}$$

$$z_l = \zeta - \int_{\theta_l}^{\theta} \frac{\boldsymbol{B}\cdot\nabla\zeta}{\boldsymbol{B}\cdot\nabla\theta} d\theta. \tag{8}$$

The way that the different local coordinate systems are connected together is discussed below in section 2.3. Once the coordinate system is defined, the geometrical quantities required to evaluate differential operators in terms of coordinate derivatives are defined in the usual way, such as the reciprocal basis vectors ∇x^i (where $i \in \{x, y, z\}$, $x^x = x$, $x^y = y$, $x^z = z$), the Jacobian $\boldsymbol{J} = (\nabla x \cdot \nabla y \times \nabla z)^{-1}$, basis vectors $\boldsymbol{e}_i = \sum_{j,k} \boldsymbol{J}\epsilon_{ijk}\nabla x^j \times \nabla x^k$ (where ϵ_{ijk} is an alternating symbol, or Levi-Civita symbol, with $\epsilon_{xyz} = 1$), contravariant components of the metric tensor $g^{ij} = \nabla x^i \cdot \nabla x^j$, and covariant components of the metric tensor $g_{ij} = \boldsymbol{e}_i \cdot \boldsymbol{e}_j$. These geometrical quantities are calculated by the grid generator and provided to BOUT++ as input in a grid file.

Many codes built with BOUT++ assume that the x- and y- coordinates are orthogonal, $\boldsymbol{e}_x \cdot \boldsymbol{e}_y = 0$ as this simplifies some differential operators. This is not required by BOUT++ or the grid generator hypnotoad [58], but is supported as the default option.

Note that in these field aligned coordinates, the y and z coordinates are very much not orthogonal, $\boldsymbol{e}_y \cdot \boldsymbol{e}_z \neq 0$, as the basis vector in the y-direction is parallel to the magnetic field $\boldsymbol{e}_y \propto \boldsymbol{b}$ while the basis vector in the z-direction points in the toroidal direction $\boldsymbol{e}_z \propto \nabla\zeta$. At X-points the coordinate system is singular (as the poloidal magnetic field vanishes there, so $\boldsymbol{B}\cdot\nabla\theta = 0$) and $\boldsymbol{e}_y$ and $\boldsymbol{e}_z$ are parallel. The grid is constructed so that X-points are at cell corners, where no quantity is evaluated, in order to avoid this singularity; nevertheless metric components may take extremely large or small values on grid points close to the X-points, which has the potential to compromise the accuracy or stability of simulations in this region.

2.2. Staggered grids

To construct staggered finite difference stencils, variables may be defined at different locations within a grid cell, either at the centre (the default) or one of the cell faces. BOUT++ represents these different locations with an '`enum class CELL_LOC`' type, whose values can be `CELL_CENTRE` for the cell centres, or `CELL_XLOW`, `CELL_YLOW`, or `CELL_ZLOW` for the cell faces in the x-, y- or z-dimensions respectively. A `Field` object has a `location` member variable of type `CELL_LOC` specifying the location within a grid cell where its values are defined. Differential operators have a '`CELL_LOC outloc`' argument to specify where within a grid cell the result should be calculated; the combination of `outloc` and the `location` of each `Field` argument to the differential operator determines the staggered or unstaggered finite difference stencil to be used for that operator.

When computing operations on staggered grids, it is necessary to evaluate geometrical quantities, such as metric coefficients, on the staggered grids, usually at the location of the output of the operation. Before version 4, support for staggered grids in BOUT++ was experimental and geometrical quantities were only available at cell centre locations. This caused an inconsistency which in principle limited the convergence to first order in the grid spacing, regardless of the numerical scheme being used. In practice the effect of the inconsistency was probably limited, since the plasma variables generally have short length scale fluctuations with much steeper gradients than the gradients of the magnetic equilibrium that determines the geometrical quantities. Simulations using slab geometry have constant geometrical coefficients and were therefore unaffected. In the version 4 series of releases the handling of geometrical quantities was refactored so that they could be provided at the cell face locations, as we now describe.

```
#include <bout/physicsmodel.hxx>

class SimpleWaveStaggered : public PhysicsModel {

  int init(bool UNUSED(restarting)) {

    g.setLocation(CELL_YLOW);

    solver->add(f, "f");
    solver->add(g, "g");

    return 0;
  }

  int rhs(BoutReal UNUSED(t)) {

    mesh->communicate(f, g);

    ddt(f) = Grad_par(g, CELL_CENTRE);
    ddt(g) = Grad_par(f, CELL_YLOW);

    return 0;
  }

  Field3D f;
  Field3D g;
};

BOUTMAIN(SimpleWaveStaggered);
```

Listing 2: BOUT++ program implementing wave equations with a staggered grid.

Before version 4, the `Mesh` class in BOUT++ handled both the logical structure of the grid (communication, etc.) and the geometrical quantities (metric, Jacobian, etc.). In version 4.0.0 [55] the geometrical quantities were moved into a separate `Coordinates` class, and from version 4.2.0 [59] a new `Coordinates` object is created for each cell location being used. When the grid file provides only cell centre information, the geometrical quantities at the cell face locations are computed by interpolating from the cell centres. From version 4.3.0 [60] it is also possible to read geometrical quantities at cell faces from the grid file, if they are present. A new grid generator, hypnotoad version 2 [58], which creates grid files including geometrical quantities at cell faces, was written from scratch in Python (replacing the older IDL hypnotoad). With these upgrades, the handling of staggered variables is consistent and evaluation of geometrical quantities on staggered grids does not limit the convergence order of the numerical schemes in BOUT++, as long as the geometrical quantities are calculated accurately enough by the grid generator. Version 4.3.0 also made the `ParallelTransform` subclasses which implement parallel derivatives (discussed in section 2.3) aware of cell location, so that they could correctly handle staggered variables.

As an illustration, our example program from Listing 1 can be extended to use staggered grids by changing the location of `g` and passing an output location to the two `Grad_par()` calls, as shown in Listing 2.

2.3. Field aligned derivatives

The extreme anisotropy between parallel and perpendicular transport in magnetised plasmas makes a grid aligned to the magnetic field very beneficial numerically. Since parallel derivatives are small, an aligned grid allows the grid spacing in one dimension to be significantly increased, reducing memory usage and relaxing time step constraints [56,61]. However magnetic shear is always present in realistic tokamak configurations and presents problems when calculating radial derivatives using a globally field aligned grid [61,62], at least when us-

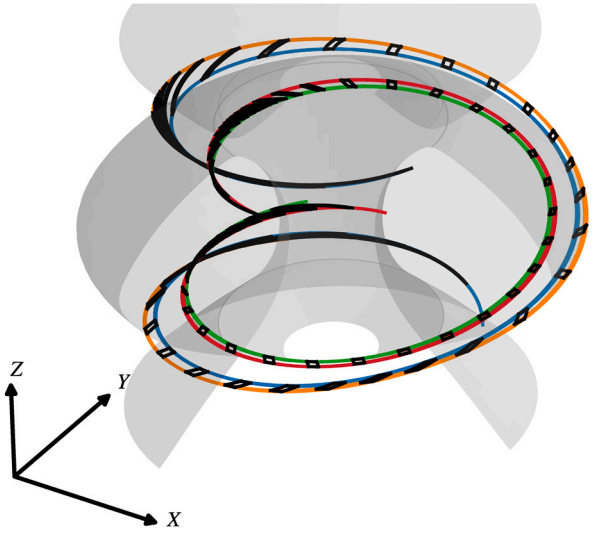


Fig. 4. Two flux tubes formed by extending a grid cell (rectangular at the outboard mid-plane) along the magnetic field. Black squares show the grid cell at different values of the poloidal/parallel coordinate y. The flux tube in the SOL has the field lines through its inner and outer corners shown in blue and orange, respectively, while green and red are the same for the flux tube in the core region. The flux tubes are plotted only until they approach near the X-points, as after that the shear would be too extreme to distinguish the grid cells in the figure. Flux tubes that are closer to the separatrix than those shown would shear notably more strongly as they approach near the X-points; the flux tubes shown are further from the separatrix than the grid for a SOL turbulence simulation would typically extend as their more moderate shear makes the figure clearer. The grey surfaces show the separatrix of this equilibrium, with a segment removed for clarity. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

ing non-spectral methods. The perpendicular planes in a globally aligned grid will be strongly distorted in some regions, as illustrated in Fig. 4, so that the 'radial' direction on the grid deviates by a large angle from the flux surface normal. There is a tension between parallel derivatives, where the most natural grid is field aligned (called an 'aligned grid' from now on), and radial derivatives, where the most natural grid follows the normals to flux surfaces (called a 'toroidal grid' from now on).

Several methods are or have been available in BOUT++ to resolve this tension. The methods discussed in this paper exploit the toroidal symmetry of the equilibrium, which allows FFT based methods to be used for very efficient interpolation in the toroidal direction. An alternative, the flux coordinate independent (FCI) method [24], which has also been implemented in BOUT++, is similar in approach to the 'shifted metric' method described in section 2.3.1, but does not align its grid to flux surfaces so that the interpolation is two dimensional. FCI methods usually choose the toroidal angle as the parallel coordinate, and construct a grid on poloidal planes for the two 'perpendicular' directions, which results in a quite different implementation from the methods discussed below. The additional flexibility of the FCI approach enables support for three dimensional geometries such as stellarators as well as avoiding discretisation issues near X-points, and the BOUT++ implementation is discussed in detail elsewhere [25,26].

In version 4 and above, the standard representation of variables in BOUT++ is on the toroidal grid, so operations involving radial derivatives are straightforward. Parallel derivatives require special handling, which is provided by various implementations of the `ParallelTransform` abstract base class. For slab-like simulations `ParallelTransformIdentity` can be used, which assumes the standard grid is itself field aligned so that parallel derivative stencils can be applied on it directly, without any 'transform'. There is an `FCI` implementation, which as mentioned above is discussed elsewhere. The `ShiftedMetric` implementation supports the 'shifted metric' and 'aligned transform' schemes which we now describe.

2.3.1. Shifted metric

The 'shifted metric' procedure introduced by Scott (2001) [62] uses a field aligned coordinate system defined locally around the x-z grid plane at each poloidal position (planes of constant y in the coordinates used here), as described in section 2.1. The local coordinate system is defined so that on each x-z grid plane, the radial x and toroidal z directions are orthogonal. The poloidal y coordinate is field aligned, but the grid points at different poloidal positions are defined in different local coordinate systems. When a parallel derivative needs to be calculated at a grid point, the magnetic field is followed to the adjacent poloidal positions, where it will intersect the x-z plane at points that are in general not grid points. The radial coordinate x is the poloidal magnetic flux function, so the planes of constant x are the flux surfaces on which magnetic field lines lie and the field line does not change its position in x; the field line is offset from the grid only in the z-direction, as shown in Fig. 5. Interpolation must be used to calculate the values of the variable at the points on the magnetic field line in order for the derivative to be calculated. BOUT++ takes advantage of the periodicity and symmetry of the grid in the toroidal z-direction to use FFT based methods for the interpolation.

The shifted metric approach, in the form just described, was introduced to BOUT++ in version 4.0.0 [55]. The `ShiftedMetric` subclass of `ParallelTransform` handles the necessary interpolation. The toroidal displacement of magnetic field lines relative to an arbitrary reference poloidal location is calculated by the grid generator and stored in an array called `zShift` to be used by `ShiftedMetric`.

BOUT++ refers to the set of interpolated values of a `Field3D` that are offset by a certain number of grid points along the magnetic field from the toroidal grid as a 'parallel slice'; in the locally field aligned coordinate system associated with each grid point (for example the central, red cross in Fig. 5), each point along the magnetic field (red asterisks in Fig. 5) comes from a separate parallel slice. Each parallel slice is a three dimensional array of values, one for each point on the toroidal grid, and is stored as a `Field3D` object. In version 4.0.0 only two parallel slices were calculated, one in each direction along the magnetic field; from version 4.3.0 it is possible to calculate an arbitrary number, allowing the use of derivatives with larger stencils (although at present BOUT++ only allows derivatives with up to two points either side of the central point in the stencil). The number of parallel slices calculated in each direction along the magnetic field is the same as the number of guard cells used in the y-direction, `MYG`.

Staggered grids in the y-direction are not currently supported when using shifted metric parallel derivatives. As shown in Fig. 5, to allow parallel derivatives to be calculated on the staggered grid from a cell centre variable, or to interpolate the variable between centred and staggered grids, an extra set of interpolated values (the purple triangles in the figure) would have to be calculated and cached. The extra computational cost and implementation complexity mean that the aligned transform approach described in section 2.3.2 is recommended when using staggered grids.

The implementation of shifted metric derivatives is mostly hidden when writing implementations of `PhysicsModel`. When `ShiftedMetric` is in use, calling `mesh->communicate(f)` calculates the parallel slices and caches them in two `std::vector<Field3D>` private members of `f`, one for 'y-up' points 'up' the magnetic field from each grid point, and one for 'y-down' points 'down' the magnetic field. Differential operators automatically use these cached parallel slices to populate the stencils for y-derivatives. If no parallel derivatives of some variable are required, it is possible as an optimisation to skip calculating the parallel slices by using the method `mesh->communicateXZ(f)`; as the name suggests, this method is intended for cases where communication is required only in the x-z plane (at present this method does still swap guard cells in the y-direction, but this is an implementation detail which may change in future).

Parallel boundary conditions can be applied in two ways. The method most suited for simulations using the shifted metric method is

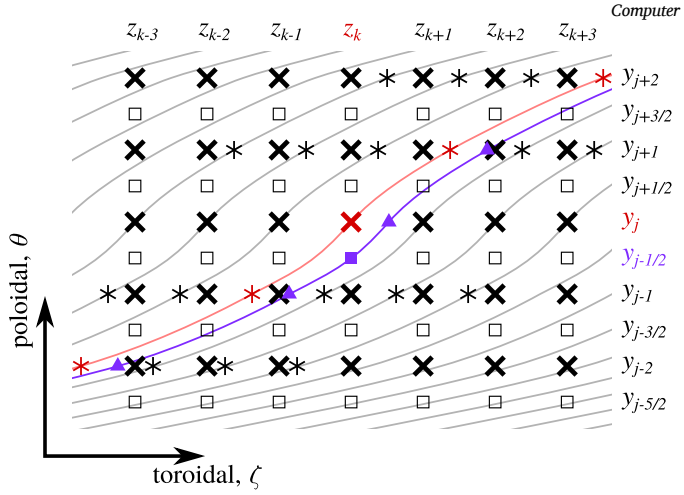


Fig. 5. Sketch to illustrate the interpolations needed for the shifted metric procedure. The grid shown is a zoom in on part of a flux surface (a surface at constant x) around some arbitrary grid point (y_j, z_k) which is a cell centre point shown by the red cross ✖. The other cell centre grid points are shown as black crosses ✖. Magnetic field lines through the centre row of grid points are grey lines, with the line through (y_j, z_k) highlighted in red. Interpolated points at $y_{j\pm1}$, $y_{j\pm2}$, shown as red stars ✳, are needed to calculate parallel derivatives at (y_j, z_k). The corresponding interpolated points for other grid points on the y_j row are shown as black stars ✳. y-staggered grid points at $y_{j-1/2}$, etc. are shown as squares □, with a chosen staggered point $(y_{j-1/2}, z_k)$ highlighted in purple ■. The magnetic field line through the staggered point $(y_{j-1/2}, z_k)$ is shown in purple, and interpolated points needed to evaluate a derivative at $(y_{j-1/2}, z_k)$ from input values on the cell centre grid are shown as purple triangles ▲.

to apply the boundary condition directly to the cached parallel slices, although only a subset of the BOUT++ boundary conditions are currently supported. The full range of boundary conditions provided by BOUT++ can be applied by transforming a variable to the aligned grid, applying the boundary condition, then transforming back to the toroidal grid; this is more expensive than the first method described because extra FFT interpolations are required.

A reasonable way to calculate the cost of the parallel derivative schemes is to count the number of FFT interpolations required, as they are expected to be the most computationally expensive part of the calculation. Also, the computational cost of applying finite difference stencils should be similar between different methods, although it is possible that there could be differences due to the different memory access patterns causing, for example, changes in vectorisation or cache use. For the shifted metric approach, the number of FFT interpolations per variable is the number of parallel slices that are needed, which is 2×`MYG`. If a y-staggered grid were supported a second set of parallel slices would be needed, corresponding to the purple triangles in Fig. 5, so twice as many FFT interpolations would be needed, 4×`MYG` per variable.

2.3.2. *Aligned transform*

The 'aligned transform' scheme described in this section is a variation of the shifted metric procedure. Variables are represented on the toroidal grid using a set of locally field aligned coordinate systems in the same way. The difference is in how the parallel derivatives are calculated. For the aligned transform scheme, a variable is transformed from the toroidal grid to the globally field aligned grid, using FFT interpolations. Parallel derivatives or interpolations are calculated on the aligned grid, as sketched in Fig. 6, and the results of these operations are transformed back to the toroidal grid.

As mentioned in section 2.3.1, using the poloidal flux function ψ for the radial coordinate means each magnetic field line, as it lies on a flux surface, has a single value of $x = \psi$ and so interpolation is only needed in the z-direction to shift from the toroidal grid to the aligned grid. `zShift` contains the toroidal angular position of a field line relative to a reference poloidal location (equal to $\int_{\theta_0}^{\theta} \frac{\boldsymbol{B}\cdot\nabla\zeta}{\boldsymbol{B}\cdot\nabla\theta} d\theta$, see equation (5)). To interpolate a `Field3D` variable $f(x, y, z)$ from the toroidal to the aligned grid we Fourier transform the z-dimension to wavenumber k_z space, giving $f_k(x, y, k_z)$, then multiply by a phase $\exp(ik_z$`zShift`$)$, and inverse Fourier transform back to a spatial grid giving $f(x, y, z+$`zShift`$)$ which gives the values of f on the aligned grid. To transform back from the aligned to the toroidal grid, we follow the same process but shifting in the opposite direction using $-$`zShift` instead of `zShift`.

Using a grid aligned to the magnetic field when computing parallel derivatives or interpolations has several advantages. Using a y-staggered grid is simple to implement, avoiding the complications associated with the shifted metric approach described in section 2.3.1, and the aligned transform scheme is computationally cheaper for staggered grids, as shown below. It is also straightforward to implement conservative numerical schemes, since each cell face through which the parallel fluxes pass is perfectly aligned with the face of a single adjacent cell, unlike in the shifted metric scheme where each of those cell faces would have a partial overlap with the faces of two neighbouring cells; standard flux-calculation methods can therefore be used, while the FFT interpolation used to transform between toroidal and aligned grids preserves the toroidal average, and therefore toroidally integrated conservation, to machine precision.

The aligned transform scheme has been implemented by adding extra functionality to the `ShiftedMetric` implementation of `ParallelTransform`; this was the most efficient approach due to the many commonalities between the aligned transform and shifted metric schemes, and because the `ShiftedMetric` class was already implemented.

Aligned transform derivatives were available in BOUT++ version 4.0.0, but in that version were intended primarily as a fall back for cases when the argument to a differential operator had no precalculated parallel slices. For example when computing `Grad_par(A+B)`, even if both `A` and `B` have cached parallel slices, the operation `(A+B)` returns a result which does not have parallel slices. In that version, the parallel slices were calculated as part of the communication routine, while transforming to and from the aligned grid did not and does not trigger communication. To avoid unnecessary communication, the aligned transform method was used rather than calculating parallel slices within the parallel derivative operator.[3] Full support for the aligned transform scheme was added in BOUT++ version 4.3.0, when an option was added to disable calculation of parallel slices by the `ShiftedMetric` class, so

[3] From version 4.2.0 it was possible to get the `ParallelTransform` instance using public methods, and therefore to calculate parallel slices without communicating, and from version 4.3.0 this was simplified by providing a method `Field3D::calcParallelSlices()`.

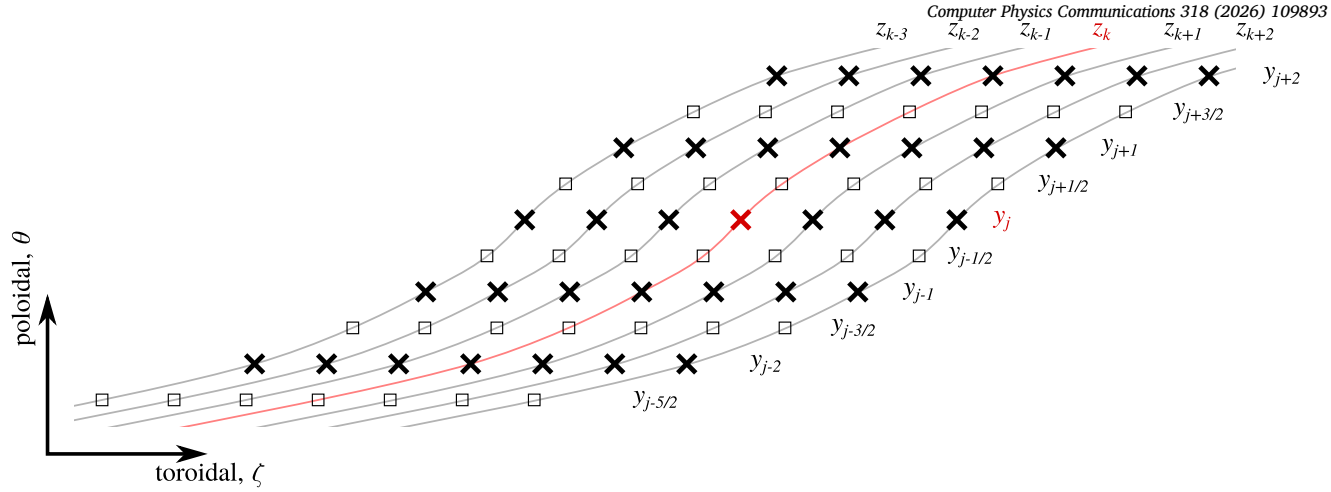


Fig. 6. Sketch of globally field aligned grid. Cell centre ✕ and y-staggered □ grid points correspond to the toroidal grid shown in Fig. 5. The central grid point is highlighted in red ✕. The magnetic field is shown as grey lines, with the line through the central grid point highlighted in red. Parallel derivatives at points on both central and staggered grids can be calculated using values on the globally aligned grid.

that all parallel operations are done by transforming variables to the aligned grid. In 4.3.0 the operation of the parallel derivative and interpolation operators was clarified by making them return their output on the same grid (aligned or toroidal) as the input argument; previously even if a field aligned input was given, the output would have been on the toroidal grid.

The aligned transform scheme can be enabled by using `ShiftedMetric` with calculation of parallel slices disabled, with no special modifications to `PhysicsModel` code. In this case the transformation to and from the field aligned grid is implicit, and is applied automatically within each parallel derivative operator. Boundary conditions are applied similarly, by transforming to the aligned grid, applying the boundary condition, and then transforming back to the toroidal grid.

The implementation just described can lead to variables being transformed from the toroidal to the aligned grid multiple times, if more than one parallel derivative or interpolation is calculated for each variable, and also when applying parallel boundary conditions. For a code using y-staggered grids, the inefficiency can be significant, as typically at least one parallel derivative, plus an interpolation between cell centre and cell face grids, are needed for each variable. A more optimised implementation can be achieved, at the cost of extra variables and lines of code in the `PhysicsModel` implementation, by defining a separate `Field3D` object to hold the transformation of each variable to the aligned grid, which can be calculated once, have boundary conditions applied, and be passed to several operations. This is the approach taken in STORM. This optimised approach can be applied to the example wave equation program, using staggered grids, as shown in Listing 3.

The optimised version of the aligned transform scheme requires one FFT interpolation per variable to calculate the aligned version, plus one FFT interpolation per parallel operation to transform the result back to the toroidal grid, so in total $1 + N_{\text{par-op}}$ interpolations per variable, where $N_{\text{par-op}}$ is the average number of parallel operations per variable. The number of parallel operations for each variable depends on the particular set of model equations being solved. For a staggered grid code $N_{\text{par-op}} \sim 2 - 3$ would be typical, allowing for one parallel interpolation between centred and staggered grids plus one or two derivative operations for the parallel first and second derivative.

Which of the shifted metric or aligned transform methods has lower computational cost depends on the situation. When using second order central finite difference methods, which have only three points in their stencils, on an unstaggered grid for parallel derivatives, only one guard cell in the y-direction is needed, `MYG` $= 1$, so the shifted metric method requires (see section 2.3.1) only $2\times$ `MYG` $= 2$ FFT interpolations, while aligned transform requires $1 + N_{\text{par-op}} = 2$ FFT interpolations if only one of the first and second derivatives must be calculated ($N_{\text{par-op}} = 1$), but three FFT interpolations for any variable where both first and second derivatives are needed ($N_{\text{par-op}} = 2$). When using staggered grids, a four point stencil is used to interpolate between centred and staggered grids, requiring `MYG` $= 2$ for the shifted metric method as well as FFT interpolation to different points for outputs on centred and staggered grids, giving a total of $4\times$ `MYG` $= 8$ FFT interpolations per variable. For the aligned transform method, since a parallel interpolation is typically required for each variable, there are likely to be two or three operations per variable $N_{\text{par-op}} = 2, 3$, so only $1 + N_{\text{par-op}} = 3, 4$ FFT interpolations per variable are needed, making the aligned transform significantly cheaper for the y-staggered grid case.

2.3.3. BOUT++ version 3

In version 3 and earlier of BOUT++ the standard grid was the aligned grid. When calculating radial derivatives or applying radial boundary conditions, variables were shifted to the toroidal grid using FFT interpolation in the toroidal direction [21], following on from one of the schemes described in Dimits (1993) [61]. The aligned transform method, section 2.3.2, is equivalent to this scheme; the differences are in implementation, due to the different choices of standard grid – toroidal vs. aligned.

3. STORM

In this section we describe the STORM code. We begin in section 3.1 by giving the model equations in SI units (the normalised form used internally by the code is given in Appendix B). The numerical methods used are described in section 3.2, the options used to initialise various types of simulations in section 3.3, and the source terms used to drive transport in section 3.4. Finally some routines for providing high time resolution output suitable for synthetic Langmuir probe diagnostics are described in section 3.5. Additional information on parallel scaling performance, post-processing tools and provenance tracking features is provided in Appendix D, Appendix E, and Appendix F.

3.1. Model

STORM solves drift reduced, cold ion, collisional fluid equations for a hydrogenic plasma [3,4], which written in SI units are

$$\frac{\partial n}{\partial t} = -\frac{1}{B}\boldsymbol{b}\cdot\nabla\phi\times\nabla n - B\nabla_{\parallel}\left(\frac{1}{B}nV_{e\parallel}\right) - nC(e\phi) + C(p_e) + \nabla_{\perp}\cdot\left(\mu_n\nabla_{\perp}n\right) + S_n, \quad \text{(electron continuity)} \quad (9)$$

```
#include <bout/physicsmodel.hxx>
class SimpleWaveAlignedTransform : public PhysicsModel {

  int init(bool UNUSED(restarting)) {

    g.setLocation(CELL_YLOW);
    g_aligned.setLocation(CELL_YLOW);

    solver->add(f, "f");
    solver->add(g, "g");

    f_aligned.setBoundary("f_aligned");
    g_aligned.setBoundary("g_aligned");

    return 0;
  }

  int rhs(BoutReal UNUSED(t)) {

    mesh->communicate(f, g);

    f_aligned = toFieldAligned(f);
    f_aligned.applyBoundary();

    g_aligned = toFieldAligned(g);
    g_aligned.applyBoundary();

    ddt(f) = fromFieldAligned(Grad_par(g_aligned, CELL_CENTRE));
    ddt(g) = fromFieldAligned(Grad_par(f_aligned, CELL_YLOW));

    return 0;
  }

  Field3D f;
  Field3D f_aligned;
  Field3D g;
  Field3D g_aligned;
};

BOUTMAIN(SimpleWaveAlignedTransform);
```

Listing 3: BOUT++ program implementing wave equations using the optimised version of the aligned transform method for parallel derivatives on a staggered grid.

$$\frac{\partial}{\partial t}\left(V_{i\parallel}+\frac{e}{m_i}A_\parallel\right) = -\frac{1}{B}\boldsymbol{b}\cdot\nabla\phi\times\nabla V_{i\parallel} - V_{i\parallel}\nabla_\parallel V_{i\parallel} - \frac{e}{m_i}\nabla_\parallel\phi - \frac{m_e}{m_i}0.51\frac{1}{\tau_{ei}}\left(V_{i\parallel}-V_{e\parallel}\right) + 0.71\frac{1}{m_i}\nabla_\parallel T_e + D_{V_{i\parallel}}\nabla_\perp^2 V_{i\parallel} - \frac{V_{i\parallel}S_n}{n}, \quad \text{(ion velocity)} \quad (10)$$

$$\frac{\partial}{\partial t}\left(V_{e\parallel}-\frac{e}{m_e}A_\parallel\right) = -\frac{1}{B}\boldsymbol{b}\cdot\nabla\phi\times\nabla V_{e\parallel} - V_{e\parallel}\nabla_\parallel V_{e\parallel} + \frac{e}{m_e}\nabla_\parallel\phi - \frac{1}{m_e n}\nabla_\parallel p_e + 0.51\frac{1}{\tau_{ei}}\left(V_{i\parallel}-V_{e\parallel}\right) - 0.71\frac{1}{m_e}\nabla_\parallel T_e + D_{V_{e\parallel}}\nabla_\perp^2 V_{e\parallel} - \frac{V_{e\parallel}S_n}{n}, \quad \text{(Ohm's law)} \quad (11)$$

$$\frac{\partial p_e}{\partial t} = -\frac{1}{B}\boldsymbol{b}\cdot\nabla\phi\times\nabla p_e - V_{e\parallel}\nabla_\parallel p_e - \frac{2}{3}B\nabla_\parallel\left(\frac{q_{e\parallel}}{B}\right) - \frac{2}{3}0.71n\left(V_{i\parallel}-V_{e\parallel}\right)\nabla_\parallel T_e - \frac{5}{3}p_e B\nabla_\parallel\left(\frac{V_{e\parallel}}{B}\right) + \frac{2m_e}{3}0.51\frac{n}{\tau_{ei}}\left(V_{i\parallel}-V_{e\parallel}\right)^2 + \frac{5}{3}p_e\left(-C(e\phi)+\frac{C(p_e)}{n}+C(T_e)\right) - \frac{m_e V_{e\parallel}^2}{3}C(p_e) + \frac{2}{3}\nabla_\perp\cdot\left(\kappa_{e\perp}\nabla_\perp T_e\right) + \frac{2}{3}S_E + \frac{m_e V_{e\parallel}^2}{3}S_n, \quad \text{(electron pressure)} \quad (12)$$

$$\frac{\partial\varpi}{\partial t} = -\frac{1}{B}\boldsymbol{b}\cdot\nabla\phi\times\nabla\varpi - V_{i\parallel}\nabla_\parallel\varpi + B\nabla_\parallel\left(\frac{1}{B}en\left(V_{i\parallel}-V_{e\parallel}\right)\right) + eC(p_e) + \nabla_\perp\cdot\left(\mu_\varpi\nabla_\perp\varpi\right). \quad \text{(vorticity)} \quad (13)$$

e is the proton charge, m_e is the electron mass and m_i is the ion mass, for which by default the value for Deuterium is used. The plasma variables are the electron density n, parallel flow velocities of electrons $V_{\parallel e}$ and ions $V_{\parallel i}$, and electron pressure $p_e = nT_e$ where T_e is the electron temperature. The generalised vorticity ϖ is related to the electrostatic potential ϕ by

$$\varpi = \nabla\cdot\left(\frac{m_i n_0}{B^2}\nabla_\perp\phi\right), \quad (14)$$

where, in a form of so-called 'Boussinesq approximation', a constant reference density n_0 is used, as discussed further in section 3.1.1, including the possibility to lift this approximation. Parallel and perpendicular are defined relative to the magnetic field $\boldsymbol{B} = B\boldsymbol{b}$ whose magnitude and direction are B and $\boldsymbol{b}$. The curvature operator is defined as

$$C(f) = \frac{1}{e}\nabla\times\left(\frac{\boldsymbol{b}}{B}\right)\cdot\nabla f. \quad (15)$$

The perpendicular dissipation coefficients μ_n, $\kappa_{e\perp}$, μ_ϖ, $D_{V_{i\parallel}}$ and $D_{V_{e\parallel}}$ are set to constant values for turbulence simulations, chosen such that fluctuations are dissipated before reaching the gyroradius scale. For computational efficiency, the grid scale in the perpendicular directions is chosen to be comparable to the gyroradius, so it is important for numerical stability that fluctuations are dissipated above this scale. It can then be verified that simulation results are insensitive to the particular values of the perpendicular dissipation parameters, as shown in [6]. Parameter dependent expressions for the perpendicular dissipation coefficients have been used for filament simulations, as discussed in section 3.1.3. There are volumetric sources of particles S_n and energy S_E which are used to drive turbulent simulations, discussed in section 3.4. The conductive parallel electron heat flux is

$$q_{e\parallel} = -\kappa_{e\parallel}\nabla_\parallel T_e - 0.71 n T_e \left(V_{i\parallel} - V_{e\parallel}\right), \tag{16}$$

where the parallel thermal conductivity is $\kappa_{e\parallel} = 3.16 n T_e \tau_{ei}/m_e$ and the collision times are given by $\tau_{ab} = 12\pi^{3/2}\epsilon_0^2 m_a^{1/2} T_a^{3/2}/2^{1/2} n_b e^4 \ln\Lambda$ with the vacuum permittivity ϵ_0 and cyclotron frequencies $\Omega_a = eB/m_a$; the Coulomb logarithm is evaluated using reference parameters n_0 and T_0 (see section 3.1.5) as $\ln\Lambda = 18 - \log\left(\left(n_0/10^{19}\,\mathrm{m}^{-3}\right)^{1/2}\left(T_0/1\,\mathrm{keV}\right)^{-3/2}\right)$.

$A_\parallel$ is the parallel component of the magnetic vector potential generated by parallel currents in the plasma modelled by the code, which act as a perturbation on top of the background equilibrium magnetic field generated by the external coils and the equilibrium plasma current, which is not part of the plasma modelled by the code. When running in electrostatic mode, the parallel component of the magnetic vector potential is neglected, $A_\parallel = 0$, so that the parallel momentum equations (10) and (11) evolve $V_{i\parallel}$ and $V_{e\parallel}$ directly. In electromagnetic mode, the equations are completed by relating $A_\parallel$ to the parallel current with Ampère's law

$$\frac{1}{\mu_0}\nabla_\perp^2 A_\parallel = -en\left(V_{i\parallel} - V_{e\parallel}\right), \tag{17}$$

where μ_0 is the vacuum permeability, and parallel derivatives follow the perturbed magnetic field

$$\nabla_\parallel f = \boldsymbol{b}_0\cdot\nabla f + \frac{1}{B}\boldsymbol{b}_0\cdot\nabla A_\parallel \times \nabla f, \tag{18}$$

where $\boldsymbol{b}_0$ is the unit vector along the background magnetic field, while perturbations to the magnitude B are neglected. So far it has only been possible to use the electromagnetic mode in slab geometry [42] due to numerical instabilities appearing near the X-point in tokamak configurations, but the cause of this issue continues to be investigated as it has been shown that including electromagnetic effects can allow longer time steps to be taken [63].

Cold ion, collisional fluid equations are not well suited to describe the edge, closed field line region of present day tokamak experiments, which even in low power L-mode regimes are typically hot enough to make the collisional assumption marginal, and hot ion effects are likely to be qualitatively important. Nevertheless a small closed field line region is included in tokamak simulations with STORM as a buffer where the sources of heat and, if necessary, particles representing fluxes from the core plasma can be introduced. The closed field line region allows turbulent fluctuations to develop around and across the separatrix; if the separatrix were taken as the radial boundary of the grid, the SOL turbulence would be unphysically modified, likely being suppressed by the absence of fully developed fluctuations at that boundary.

There is also an isothermal mode, where T_e is taken to be constant and the electron pressure equation (12) is neglected.

3.1.1. Boussinesq approximations

A so-called 'Boussinesq approximation', as presented in section 3.1, is normally used for numerical efficiency, where the density is replaced by a constant reference value so that it can be removed from the divergence in (14). This form of Boussinesq approximation was introduced by [12] and is now used as the standard option, as it has better conservation properties than the version originally used in STORM [33], and so avoids large, spurious current sources that could be produced by the original form in some situations. The original form of Boussinesq approximation from [33], rather than replacing the density in the full expression (21) with a constant reference value n_0, instead moved the density outside the derivative terms $\partial/\partial_t\left(\nabla\cdot\left(m_i n B^{-2}\nabla_\perp\phi\right)\right) \approx n\partial/\partial_t\left(\nabla\cdot\left(m_i B^{-2}\nabla_\perp\phi\right)\right)$, resulting in a modified generalised vorticity

$$\varpi_{\mathrm{mod}} = \nabla\cdot\left(m_i B^{-2}\nabla_\perp\phi\right) \tag{19}$$

and the vorticity equation

$$\begin{aligned}\frac{\partial\varpi_{\mathrm{mod}}}{\partial t} = &-\frac{1}{B}\boldsymbol{b}\cdot\nabla\phi\times\nabla\varpi_{\mathrm{mod}} - V_{i\parallel}\nabla_\parallel\varpi_{\mathrm{mod}}\\ &+\frac{1}{n}B\nabla_\parallel\left(\frac{1}{B}en\left(V_{i\parallel}-V_{e\parallel}\right)\right) + e\frac{C(p_e)}{n}\\ &+\nabla_\perp\cdot\left(\mu_\varpi\nabla_\perp\varpi_{\mathrm{mod}}\right).\end{aligned} \tag{20}$$

It is also possible to run without any Boussinesq approximation, retaining the full density in the generalised vorticity, giving [39,64]

$$\varpi_{\mathrm{full}} = \nabla\cdot\left(\frac{m_i n}{B^2}\nabla_\perp\phi\right) \tag{21}$$

$$\begin{aligned}\frac{\partial\varpi_{\mathrm{full}}}{\partial t} = &-\frac{1}{B}\boldsymbol{b}\cdot\nabla\phi\times\nabla\varpi_{\mathrm{full}} - \frac{1}{2}\boldsymbol{b}\cdot\nabla\left(\left|\frac{\boldsymbol{b}\times\nabla\phi}{B}\right|^2\right)\times\nabla n\\ &- V_{i\parallel}\nabla_\parallel\varpi_{\mathrm{full}} + B\nabla_\parallel\left(\frac{1}{B}en\left(V_{i\parallel}-V_{e\parallel}\right)\right)\\ &+eC(p_e) + \nabla_\perp\cdot\left(\mu_\varpi\nabla_\perp\varpi_{\mathrm{full}}\right).\end{aligned} \tag{22}$$

The options for numerical methods that allow lifting the Boussinesq approximation are discussed in section 3.2.2.

3.1.2. Boundary conditions

The radial boundaries are essentially dissipative numerical buffers and are discussed in section 3.2.3.

The parallel boundary conditions for the plasma are set by the physics of the Debye sheath, with Bohm boundary conditions [65] being imposed at the sheath entrance location (the non-quasineutral sheath itself is not included in the simulation domain). These are outflow boundary conditions where no explicit boundary condition is imposed on scalar variables (density and pressure), while Dirichlet-type boundary conditions are imposed on the parallel fluxes of particles and energy as follows. The ion parallel velocity at the sheath entrance is set greater than or equal to the sound speed

$$\pm V_{i\parallel}\Big|_{\mathrm{sheath}} \geq \sqrt{\frac{T_e|_{\mathrm{sheath}}}{m_i + m_e}} \tag{23}$$

and the electron parallel velocity is regulated by the electrostatic potential at the sheath entrance

$$\pm V_{e\parallel}\Big|_{\mathrm{sheath}} = \sqrt{\frac{m_i\, T_e|_{\mathrm{sheath}}}{2\pi m_e\left(m_i + m_e\right)}}\exp\left(-\frac{e\max\left(\phi|_{\mathrm{sheath}}, 0\right)}{T_e|_{\mathrm{sheath}}}\right), \tag{24}$$

where the electrostatic potential at the wall is taken to be $\phi|_{\mathrm{wall}} = 0$. Negative values of the sheath potential are disregarded to avoid unphysically large values of $V_{e\parallel}$ which might occur in rare, exceptionally large fluctuations; (24) is derived assuming an electron repelling sheath and so is not valid for $\phi|_{\mathrm{sheath}} \lesssim 0$. The total electron parallel energy flux at the sheath entrance is

$$\begin{aligned}\pm Q_{e\parallel}\Big|_{\mathrm{sheath}} &= \left(e\,\phi|_{\mathrm{sheath}} + 2\,T_e|_{\mathrm{sheath}}\right) n|_{\mathrm{sheath}}\, V_{e\parallel}\Big|_{\mathrm{sheath}}\\ &\approx \left(0.5\ln\left(\frac{m_i}{2\pi m_e}\right) + 2\right) n|_{\mathrm{sheath}}\, T_e|_{\mathrm{sheath}}\, V_{e\parallel}\Big|_{\mathrm{sheath}}\\ &\approx 5.18\, n|_{\mathrm{sheath}}\, T_e|_{\mathrm{sheath}}\, V_{e\parallel}\Big|_{\mathrm{sheath}}.\end{aligned} \tag{25}$$

The signs in (23), (24), and (25) are such that the parallel velocities and parallel energy flux are directed outwards at each boundary. Note that in the second line (25), while the time-varying $\phi|_{\text{sheath}}$ is used to calculate $V_{e\parallel}\Big|_{\text{sheath}}$ using (24), in the prefactor $\phi|_{\text{sheath}}$ is replaced by the floating potential $0.5\ln\left(m_i/2\pi m_e\right)T_e|_{\text{sheath}}$. It would be possible to use $\phi|_{\text{sheath}}$ also in the prefactor, but (25) is the form that has been used for previous publications with STORM.

3.1.3. Perpendicular dissipation parameters used for filament simulations

For turbulence simulations, the perpendicular dissipation parameters are set to constant values, as discussed in section 3.1. Filament simulations have used expressions that depend on the evolving n_e and T_e,

$$\mu_n = \left(1+1.3q^2\right)\left(1+\frac{T_i}{T_e}\right)\frac{T_e}{m_e\Omega_e^2\tau_{ei}} \tag{26}$$

$$\kappa_{e\perp} = 4.66\left(1+1.6q^2\right)\frac{n_eT_e}{m_e\Omega_e^2\tau_{ei}} \tag{27}$$

$$\mu_\varpi = \frac{3}{4}\left(1+1.6q^2\right)\frac{T_i}{m_i\Omega_i^2\tau_{ii}}, \tag{28}$$

which are the classical values of Braginskii when $q=0$. μ_n and μ_ϖ are evaluated with a finite ion temperature $T_i = T_e$, despite the model assuming cold ions, in order to provide finite perpendicular viscosity in the vorticity equation in particular. The velocity diffusion coefficients $D_{V_{i\parallel}}$ and $D_{V_{e\parallel}}$ are set to zero for these filament simulations. The enhancement of the perpendicular dissipation coefficients for $q>0$ was originally motivated by analogy with the neoclassical radial diffusion coefficients on closed flux surfaces [66], with q nominally the safety factor, but can better be viewed as an ad-hoc enhancement to prevent gyroradius scale gradients within a cold ion model and to improve numerical stability, with q an arbitrary parameter. Values for q have been chosen, as relevant for a tokamak edge, for filament simulations between 4.8 [38] and 7 [33,37], enhancing the dissipation coefficients by factors between 30 and 80 compared to the classical values.

3.1.4. Coordinates and geometry

For simulations in tokamak geometry, the usual BOUT++ locally field aligned coordinate system discussed in section 2.1 is used, with orthogonal x- and y-coordinates. For curved slab geometries $\{x,y,z\}$ are simply Cartesian coordinates, with x being radial, y the coordinate along magnetic field lines and z binormal. The metric is then simply the identity matrix.

3.1.5. Normalisations

Internally, dimensionless variables are used which are normalised to have magnitudes of order unity in order to minimise the possibility of underflow or overflow errors. Writing the normalised version of a variable or operator f as $\hat{f}$, reference values of density n_0, temperature T_0, and magnetic field B_0, for which suitable values are chosen depending on the parameters of a given simulation, are used to normalise the density $\hat{n} = n/n_0$, electron temperature $\hat{T}_e = T_e/T_0$, and magnetic field $\hat{B} = B/B_0$. The ion mass is used as the reference value, so the normalised electron mass is $\hat{m}_e = m_e/m_i$. Bohm normalisation is used for lengths and times, $\hat{\nabla} = \rho_{s0}\nabla$ and $\partial/\partial\hat{t} = \Omega_{i0}^{-1}\partial/\partial t$, with the reference sound Larmor radius $\rho_{s0} = c_{s0}/\Omega_{i0}$, cold ion sound speed $c_{s0} = \sqrt{T_0/m_i}$ and cyclotron frequency $\Omega_{i0} = eB_0/m_i$. The electromagnetic potentials are normalised as $\hat{\phi} = e\phi/T_0$ and $\hat{A}_\parallel = 2eA_\parallel/\beta_0 m_i c_{s0}$ where $\beta_0 = 2\mu_0 n_0 T_0/B_0^2$. The normalised vorticity is $\hat{\varpi} = \hat{\nabla}\cdot\left(\hat{B}^{-2}\hat{\nabla}_\perp\hat{\phi}\right) = \varpi/en_0$ for the 'standard' Boussinesq approximation (14); $\hat{\varpi}_{\text{full}} = \hat{\nabla}\cdot\left(\hat{n}\hat{B}^{-2}\hat{\nabla}_\perp\hat{\phi}\right) = \varpi_{\text{full}}/en_0$ for the full, non-Boussinesq version (21); and $\hat{\varpi}_{\text{mod}} = \hat{\nabla}\cdot\left(\hat{B}^{-2}\hat{\nabla}_\perp\hat{\phi}\right) = \varpi_{\text{mod}}/e$ for the 'original STORM' Boussinesq approximation (19). The source terms are normalised as $\hat{S}_n = S_n/n_0\Omega_{i0}$ and $\hat{S}_E = S_E/n_0T_0\Omega_{i0}$. Normalisations of the parameters and model equations follow from these definitions, as detailed in Appendix B.

In tokamak geometries, the y and z coordinates are dimensionless (angle-like) and so do not need to be normalised, while $x=\psi$ is normalised as $\hat{x} = x/\rho_{s0}^2B_0$. As the reciprocal metric components are by definition $g^{ij} = \nabla x^i\cdot\nabla x^j$ they are normalised as $\hat{g}^{xx} = g^{xx}/\rho_{s0}^2B_0^2$, $\hat{g}^{yy} = g^{yy}\rho_{s0}^2$, $\hat{g}^{zz} = g^{zz}\rho_{s0}^2$, $\hat{g}^{xy} = g^{xy}/B_0$, $\hat{g}^{xz} = g^{xz}/B_0$, and $\hat{g}^{yz} = g^{yz}\rho_{s0}^2$, while the metric components are the inverse $\hat{g}_{xx} = g_{xx}\rho_{s0}^2B_0^2$, $\hat{g}_{yy} = g_{yy}/\rho_{s0}^2$, $\hat{g}_{zz} = g_{zz}/\rho_{s0}^2$, $\hat{g}_{xy} = g_{xy}B_0$, $\hat{g}_{xz} = g_{xz}B_0$, and $\hat{g}_{yz} = g_{yz}/\rho_{s0}^2$. The Jacobian $J = (\nabla x\cdot\nabla y\times\nabla z)^{-1}$ is normalised as $\hat{J} = JB_0/\rho_{s0}$.

In slab geometries all the coordinates are normalised by ρ_{s0}, so $\hat{x} = x/\rho_{s0}$, $\hat{y} = y/\rho_{s0}$, and $\hat{z} = z/\rho_{s0}$. The metric and Jacobian are dimensionless and therefore do not need to be normalised.

3.2. Numerical methods

The physical model described in the previous subsection must be discretised so that it can be solved numerically. We now describe in some detail the algorithms used, to provide a reference for the code.

STORM uses finite difference methods, except for a few places noted below where FFTs are used for toroidal derivatives or interpolation. The implementation of the numerical schemes is mostly provided by the BOUT++ framework; the sheath boundary conditions require a lower level implementation, section 3.2.3. Here, we specify the methods STORM used as standard for each operation. The standard set of choices described here can be overridden at run time by options in the input file if necessary; differences applied in some particular study would be noted explicitly in the paper describing it. To ensure positivity of the density n and pressure p_e, their logarithms $\log n$ and $\log p_e$ are evolved numerically, as shown in the normalised equations in Appendix B.

As introduced above, one of the distinctive features of STORM is its use of staggered grids in the parallel direction. This helps to avoid the chequerboard instability – the numerical decoupling of 'odd' grid points from 'even' grid points – associated with advective equations [67, sections 6.2, 6.3]. The parallel direction has been found to be susceptible to the chequerboard instability while the perpendicular directions are not, likely due to the faster parallel dynamics and the presence of sheath boundary conditions that impose sharp variations in the parallel direction. The 'scalar' variables n, p_e, ϖ, ϕ are evolved or evaluated on the 'cell centre' grid, while 'vector' variables $V_{i\parallel}$, $V_{e\parallel}$, and $A_\parallel$, which represent the parallel component of a vector quantity, are evolved on a staggered grid offset to half way between the cell centres, at the 'cell faces'. This means that the evolution equations (9), (12), and (13), and therefore each component (such as derivatives) within those equations, are evaluated at cell centre grid points. (10) and (11), and each component within them, are evaluated at cell face grid points. As shown in Figs. 2 and 3, in BOUT++ the boundaries of the domain are taken to be at cell face locations.

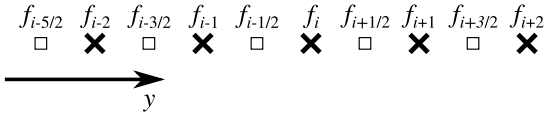
The following subsections detail the specific methods used for each type of numerical operation: parallel interpolation and the differential operators used for parallel derivatives, perpendicular derivatives, and curvature operators in section 3.2.1; Laplace inversions needed to calculate ϕ and $A_\parallel$ in section 3.2.2; boundary conditions in section 3.2.3; and the time solver in section 3.2.4.

3.2.1. Differential operators and interpolation

Differential operators in BOUT++ are defined in terms of coordinate derivatives $\partial/\partial x$, $\partial/\partial y$ and $\partial/\partial z$ in the locally field aligned coordinate system (section 2.3), combined with appropriate geometric factors – metric components, the Jacobian, etc. – to account for general curvilinear coordinates, as described in the BOUT++ documentation [53]. The grid generator hypnotoad is designed to ensure that all grid spacings are slowly and smoothly varying, so that this coordinate transformation provides an accurate representation of the differential operators [58]. In this section, except where noted otherwise, we detail just the discretisations used for the underlying coordinate derivatives in each operator, as used by STORM. The aligned transform scheme (section 2.3.2) is used,

so that parallel derivatives and interpolation are calculated on a globally field aligned grid, with the toroidal interpolations needed to transform variables onto that aligned grid being calculated using FFTs. The chain rule is used to split several operators in equations (9)-(13) as the split forms have slightly better numerical stability. The exact forms as used in the code are shown in the normalised equations in Appendix B. Within this section, we quote individual terms from the split forms, which each correspond to an individual term in equations (B.2)-(B.6), when referring to the terms that are discretised, but write them in SI units.

To describe finite difference stencils for parallel derivatives, we denote the values of some variable f at grid points using a subscript that labels the y-index of the points, with indices on the staggered grid being offset by one half, as shown in the sketch.



Derivatives parallel to the unperturbed magnetic field use second order accurate finite difference methods. Derivatives parallel to the full magnetic field are calculated, in electromagnetic mode, by adding a correction due to $A_\parallel$ discussed below. Advective derivatives, of the form $v\nabla_\parallel f$, use an upwind method, while the other parallel derivatives use a centred method. Derivatives account for whether the argument is on the same grid as the output, or on the grid staggered relative to the output, by using the appropriate stencil. For the non-advective derivatives, this is essentially just a change in grid spacing, so that unstaggered derivatives use

$$\left.\frac{\partial f}{\partial y}\right|_i = \frac{f_{i+1} - f_{i-1}}{2\Delta_{y,i}} \tag{29}$$

where $\Delta_{y,i}$ is the y-grid spacing at point i, which is used for $\nabla_\parallel T_e$ in the electron pressure equation (12) and $\nabla_\parallel \log n$ in the vorticity equation (13). Staggered derivatives use

$$\left.\frac{\partial f}{\partial y}\right|_{i-1/2} = \frac{f_i - f_{i-1}}{\Delta_{y,i-1/2}}, \tag{30}$$

which is used for: $\nabla_\parallel \left(V_{e\parallel}/B\right)$ in the continuity equation (9); $\nabla_\parallel \phi$, $\nabla_\parallel T_e$, and $\nabla_\parallel \log n$ in the ion momentum equation (10) and Ohm's law (11); $\nabla_\parallel T_e$ in the calculation of $q_{e\parallel}$ (16); $\nabla_\parallel \left(q_{e\parallel}/B\right)$ and $\nabla_\parallel \left(V_{e\parallel}/B\right)$ in the electron pressure equation (12); and $\nabla_\parallel \left(\left(V_{i\parallel} - V_{e\parallel}\right)/B\right)$ in the vorticity equation (13).

Parallel advection terms use upwind schemes. For the $V_{i\parallel}\nabla_\parallel V_{i\parallel}$ and $V_{e\parallel}\nabla_\parallel V_{e\parallel}$ terms in the ion and electron momentum equations (10) and (11), the velocity and the variable are on the same grid and the unstaggered form

$$v\left.\frac{\partial f}{\partial y}\right|_i = \begin{cases} v_i \frac{\left(\frac{3}{2}f_i - 2f_{i-1} + \frac{1}{2}f_{i-2}\right)}{\Delta_{y,i}} & v_i \geq 0 \\ -v_i \frac{\left(\frac{3}{2}f_i - 2f_{i+1} + \frac{1}{2}f_{i+2}\right)}{\Delta_{y,i}} & v_i < 0 \end{cases} \tag{31}$$

is used. The remaining parallel advection terms, $V_{e\parallel}\nabla_\parallel \log n$, $V_{e\parallel}\nabla_\parallel \log p_e$, and $V_{i\parallel}\nabla_\parallel \varpi$ in the continuity (9), electron pressure (12), and vorticity (13) equations use the form with a staggered velocity

$$\begin{aligned} v\left.\frac{\partial f}{\partial y}\right|_i &= \left.\frac{\partial vf}{\partial y}\right|_i - \left.\frac{\partial v}{\partial y}f\right|_i \\ &= \frac{\left(X_+ + X_-\right)}{\Delta_{y,i}} - \frac{\left(v_{i+1/2} - v_{i-1/2}\right)}{\Delta_{y,i}} f_i, \end{aligned} \tag{32}$$

where

$$X_+ = \begin{cases} v_{i+1/2}\left(\frac{3}{2}f_i - \frac{1}{2}f_{i-1}\right) & v_{i+1/2} >= 0 \\ v_{i+1/2}\left(\frac{3}{2}f_{i+1} - \frac{1}{2}f_{i+2}\right) & v_{i+1/2} < 0 \end{cases} \tag{33}$$

$$X_- = \begin{cases} -v_{i-1/2}\left(\frac{3}{2}f_{i-1} - \frac{1}{2}f_{i-2}\right) & v_{i-1/2} >= 0 \\ -v_{i-1/2}\left(\frac{3}{2}f_i - \frac{1}{2}f_{i+1}\right) & v_{i-1/2} < 0 \end{cases}. \tag{34}$$

Note that this form ensures that the result is continuous as either $v_{i+1/2}$ or $v_{i-1/2}$ pass through zero, which reduces the sensitivity to round off errors.

In several places scalar variables n, T_e, ϕ are required on the cell face grid in equations (10) and (11), while vector variables $V_{e\parallel}$, $V_{i\parallel}$, $A_\parallel$ are required on the cell centre grid in equations (9), (12), and (13). The values are calculated by interpolating in the parallel y direction on a field aligned grid. The interpolation is performed in grid-index space using a cubic polynomial interpolation from a four point stencil with two points on either side of the result, so that a variable f is calculated at each offset point $i - 1/2$ as

$$f_{i-1/2} = \frac{-f_{i-2} + 9f_{i-1} + 9f_i - f_{i+1}}{16}. \tag{35}$$

Note that interpolation in grid index space is equivalent to interpolation in the y-coordinate, as tokamak grids for BOUT++ always use constant grid spacing in y, with any refinement of the grid in specific locations being achieved by different choices of y-coordinate.

$E \times B$ advection and corrections to the parallel gradient due to magnetic field perturbations (18) both have the form

$$\frac{1}{B}\boldsymbol{b}_0 \cdot \nabla f \times \nabla h \tag{36}$$

and are discretised in the same way. The x and z coordinates in the locally field aligned coordinate system represent perpendicular variations, but are not in general perpendicular to the magnetic field (as discussed in section 2.1), which means that y-derivative terms contribute to the operator (36). These y-derivatives are neglected on the assumption that parallel gradients are negligible compared to perpendicular gradients, which is reasonable for turbulent fluctuations in a magnetised plasma. For structures that have a perpendicular length scale within a flux surface comparable to the parallel length scales, i.e. axisymmetric modes or those with low toroidal mode number, the neglect of y-derivatives means that the effects of poloidal $E \times B$ drift and radial drift from E_{poloidal} due to these modes are not consistently included, although some parts are captured by y-derivatives that are included in the curvature operator (39). Careful validation would be needed in situations where these kinds of poloidal $E \times B$ drift or E_{poloidal} are an important contribution. In regions where parallel gradients are extremely strong, such as near the sheath entrances at the target plates, the assumption is violated and so the effects of drifts are again not fully captured. Neglect of the y-derivatives in differential operators is consistent with their neglect when calculating the electromagnetic potentials ϕ and $A_\parallel$, see section 3.2.2; including all y-derivatives would be straightforward in the differential operators described here, but would require them also to be included in the potential solvers, which is challenging. The remaining x- and z-derivative components are calculated using a second order accurate Arakawa stencil [68]. The Clebsch property of the locally aligned coordinate system used by BOUT++ $\boldsymbol{B} = \nabla z \times \nabla x$, combined with the neglect of y-derivatives, gives a particularly simple form of this operator

$$\frac{1}{B}\boldsymbol{b}_0 \cdot \nabla f \times \nabla h \approx \frac{\partial f}{\partial z}\frac{\partial h}{\partial x} - \frac{\partial f}{\partial x}\frac{\partial h}{\partial z}, \tag{37}$$

which is discretised as

$$\left(\frac{1}{B}\boldsymbol{b}_0 \cdot \nabla f \times \nabla h\right)_{i,j} = -\frac{1}{3}\left[J^{++}_{i,j}(h,f) + J^{+\times}_{i,j}(h,f) + J^{\times+}_{i,j}(h,f)\right] \tag{38}$$

$$\begin{aligned} J^{++}_{i,j}(h,f) = \frac{1}{4\Delta_{x,i}\Delta_z}\Big[&\left(h_{i,j+1} - h_{i,j-1}\right)\left(f_{i+1,j} - f_{i-1,j}\right) \\ &- \left(h_{i+1,j} - h_{i-1,j}\right)\left(f_{i,j+1} - f_{i,j-1}\right)\Big] \end{aligned}$$

$$J^{+\times}_{i,j}(h,f) = \frac{1}{4\Delta_{x,i}\Delta_z}\Big[h_{i,j+1}\left(f_{i+1,j+1} - f_{i-1,j+1}\right)$$

$$
\begin{aligned}
&- h_{i,j-1}\left(f_{i+1,j-1} - f_{i-1,j-1}\right)\\
&- h_{i+1,j}\left(f_{i+1,j+1} - f_{i+1,j-1}\right)\\
&+ h_{i-1,j}\left(f_{i-1,j+1} - f_{i-1,j-1}\right)\Big]
\end{aligned}
$$

$$
\begin{aligned}
J^{\times+}_{i,j}(h,f) = \frac{1}{4\Delta_{x,i}\Delta_z}\Big[&h_{i+1,j+1}\left(f_{i+1,j} - f_{i,j+1}\right)\\
&- h_{i-1,j-1}\left(f_{i,j-1} - f_{i-1,j}\right)\\
&- h_{i+1,j-1}\left(f_{i+1,j} - f_{i,j-1}\right)\\
&+ h_{i-1,j+1}\left(f_{i,j+1} - f_{i-1,j}\right)\Big]
\end{aligned}
$$

where i is the x-index and j is the z-index, $\Delta_{x,i}$ is the spacing of the x-grid and Δ_z is the constant spacing of the z-grid. Note that here $\boldsymbol{b}_0$ is in the y-direction of a right handed $\{x, y, z\}$ coordinate system, so the order of indices in (38) is reversed compared to (36)-(38) of [68].

The operators used for perpendicular dissipation (section 3.1.3) also neglect y-derivatives. The diffusion terms $\nabla^2_\perp f$ use BOUT++'s `Delp2` operator, which was written to be the discrete operator whose inverse is calculated by the 'Laplace inversion' routines (section 3.2.2). `Delp2` first performs a toroidal Fourier transform of its argument, then computes x-derivatives using a second order centred finite difference method, and z-derivatives by multiplying by the appropriate power of the complex wave number for each Fourier mode; for completeness, note that `Delp2` does have an argument allowing to switch to an implementation with centred finite difference for all derivatives, but STORM does not use this. When non-constant dissipation parameters are used the diffusion operator is split in the form

$$\nabla\cdot\left(\mu\nabla_\perp f\right) = \mu\nabla^2_\perp f + \nabla_\perp\mu\cdot\nabla_\perp f$$

as shown in Appendix B, so it is necessary to compute terms of the form $\nabla_\perp\mu\cdot\nabla_\perp f$. As BOUT++ does not provide this particular operator, it is implemented in STORM in terms of coordinate derivatives as

$$\nabla_\perp\mu\cdot\nabla_\perp f = g^{xx}\frac{\partial\mu}{\partial x}\frac{\partial f}{\partial x} + g^{zz}\frac{\partial\mu}{\partial z}\frac{\partial f}{\partial z}$$

where g^{xx} and g^{zz} are contravariant components of the metric tensor and we recall that in the locally field aligned coordinate system, $g^{xz} = 0$ at the grid points as x and z are orthogonal there, section 2.3.1. The coordinate derivative $\partial/\partial x$ is discretised with a second order central finite difference method, and $\partial/\partial z$ usually uses an FFT method, but some filament simulations in slab geometry use second order central finite difference for symmetry.

Finally, we turn to the curvature operator (15)

$$
\begin{aligned}
C(f) &= \frac{1}{e}\nabla\times\left(\frac{\boldsymbol{b}}{B}\right)\cdot\nabla f\\
&= \frac{1}{e}\left[\nabla\times\left(\frac{\boldsymbol{b}}{B}\right)^x\frac{\partial f}{\partial x} + \nabla\times\left(\frac{\boldsymbol{b}}{B}\right)^y\frac{\partial f}{\partial y} + \nabla\times\left(\frac{\boldsymbol{b}}{B}\right)^z\frac{\partial f}{\partial z}\right].
\end{aligned}
\tag{39}
$$

For simulations in tokamak geometry, the contravariant components of the vector $\nabla\times(\boldsymbol{b}/B)$ are calculated by the grid generator hypnotoad [58] using derivatives of the interpolating functions used to represent the poloidal flux ψ and poloidal current function $I(\psi) = RB_{\text{toroidal}}$ of the equilibrium, where R is the major radius and B_{toroidal} is the equilibrium toroidal magnetic field. As for other operators described above, $\partial/\partial x$ and $\partial/\partial y$ use a second order centred finite difference method, and $\partial/\partial z$ uses an FFT method. For simulations in curved slab geometry, the curvature is approximated by the form for a purely toroidal magnetic field with a radius of curvature R_c that is large compared to the radial width of the simulation domain, $\nabla\times(\boldsymbol{b}/B)^z = 2/B_0R_c$ and $\nabla\times(\boldsymbol{b}/B)^x = \nabla\times(\boldsymbol{b}/B)^y = 0$, where B_0 is the constant, reference magnetic field, as shown in Appendix C; $\partial/\partial z$ is calculated in slab simulations either with an FFT, or a second order central finite difference method.

3.2.2. Laplace inversion

To calculate ϕ from ϖ or, in electromagnetic mode, to separate $A_\parallel$, $V_{e\parallel}$, and $V_{i\parallel}$ requires solving a boundary value problem involving perpendicular gradients, referred to by BOUT++ as 'Laplace inversion' because the leading term in the equations is the perpendicular Laplacian $\nabla^2_\perp$. As mentioned in section 3.2.1, there are y-derivative contributions to the perpendicular gradient, because the x- and z-coordinates are not in general perpendicular to the magnetic field. The underlying issue is not dependent on the particular choice of coordinate system however, as in the closed field line region a line following the binormal direction, perpendicular to $\boldsymbol{B}$ but within a flux surface, will usually cover the whole flux surface ergodically. Therefore the 'plane' perpendicular to $\boldsymbol{B}$ on which we must solve the boundary value problem actually fills the whole volume, making the solution three dimensional rather than two dimensional. In order to make the problem two dimensional, y-derivatives are neglected, which can be justified to the extent that parallel gradients are much smaller than perpendicular gradients, as discussed in section 3.2.1; three dimensional 'Laplace solvers' are under development in BOUT++ to remove this approximation, although these are computationally intensive and it is also challenging to formulate an appropriate parallel boundary condition.

Using either form of Boussinesq approximation, the coefficients in the equation to be solved for ϕ, (14) or (19), are independent of the toroidal coordinate z. This makes it possible to use the default BOUT++ solver (called 'cyclic'), which transforms the equation using FFTs in the toroidal coordinate z into a set of decoupled, ordinary differential equations where the independent variable is the radial x-coordinate. A second order central finite difference discretisation puts these in the form of a tridiagonal matrix, and the solution is computed by a direct method using a two level partitioning algorithm [69]. For the non-Boussinesq case, the coefficients of (21) involve the density n, which has toroidal variation and so prevents decoupling by toroidal Fourier transform. A multigrid algorithm, which discretises both x- and z-derivatives using second order central finite differences, can be used and has been successful for filament simulations in slab geometry [39,42]. For turbulent simulations in tokamak geometry, it has been observed to be more robust to use an iterative algorithm [70] called the 'Naulin Solver' where iterations ϕ^i are updated by solving

$$
\begin{aligned}
&\nabla^2_\perp\phi^{i+1} + \left\langle\frac{B^2}{n}\right\rangle_z\nabla_\perp\left\langle\frac{n}{B^2}\right\rangle_z\cdot\nabla_\perp\phi^{i+1}\\
&= \frac{B^2}{m_i n}\varpi - \left(\frac{B^2}{n}\nabla_\perp\frac{n}{B^2} - \left\langle\frac{B^2}{n}\right\rangle_z\nabla_\perp\left\langle\frac{n}{B^2}\right\rangle_z\right)\cdot\nabla_\perp\phi^i
\end{aligned}
\tag{40}
$$

with the 'cyclic' solver, where $\langle\cdot\rangle_z$ denotes a toroidal average.

Radial boundary conditions are required to solve for ϕ. The radial boundaries are located at an arbitrary flux surface where the grid ends, not a physical boundary such as a wall, and the region near the radial boundaries is a non-physical buffer zone where enhanced numerical dissipation is used to damp fluctuations, section 3.2.3. Therefore the radial boundary condition is somewhat arbitrary, and a Dirichlet boundary condition is used, with a non-zero boundary value that is updated over time. To improve numerical stability and avoid restricting the timestep, it has been found that allowing a poloidal variation in the value to which this boundary condition is set is important to avoid sharp boundary layers in the dissipative radial buffers, which would form if the boundary value were far from values consistent with parallel force balance for the electrons. For turbulent simulations, this poloidal variation is determined by a relaxation procedure. The radial boundary values are updated at intervals, whose length is chosen for each simulation to give numerically stable solutions. At the end of each interval, the boundary value is set to the time and toroidal average of ϕ at the adjacent grid point over the interval, so that the average radial profile of ϕ relaxes toward zero gradient at the boundaries. For filament simulations, which are initialised on top of a stationary background plasma (section 3.3), the background value of ϕ is used as the radial boundary condition.

For Fourier components of ϕ with low toroidal mode numbers, the approximation that parallel derivatives are much smaller than perpendicular derivatives can break down as the length scale of toroidal derivatives for these components is comparable to the major radius. Near the X-point, radial derivatives are also small due to the flux expansion there. This can result in unphysical solutions in some cases, for which a partial solution is to include y-derivatives when solving for the toroidally-constant part of ϕ (the zero mode number component of the toroidal Fourier transform) [12]. There is an option to enable this partial solution, which can be used with the standard Boussinesq approximation. When the option `split_n0=`**`true`** is set, the toroidally constant part of (14) is solved including y-derivatives. The operator is discretised with second order centred finite differences, and then solved on the full $x-y$ plane using an iterative scheme implemented using PETSc [71,72] and calling the BoomerAMG preconditioner from Hypre [73]. Radial x-boundaries use the same boundary conditions as the standard solve. Parallel/poloidal y-boundaries use Neumann boundary conditions by default, but appropriate settings here are still under investigation.

In electromagnetic mode, the variables advanced by the time solver in the ion and electron momentum equations (10) and (11) are $\chi_i = V_{i\parallel} + eA_\parallel/m_i$ and $\chi_e = V_{e\parallel} - eA_\parallel/m_e$. To obtain $A_\parallel$, $V_{i\parallel}$, and $V_{e\parallel}$ separately, we solve Ampère's law (17), in the form

$$\nabla_\perp^2 A_\parallel + \left(\frac{1}{m_e} + \frac{1}{m_i}\right) e^2 \mu_0 n A_\parallel = -e\mu_0 n \left(\chi_i - \chi_e\right), \tag{41}$$

for $A_\parallel$, which we can then use to calculate $V_{i\parallel}$ and $V_{e\parallel}$ directly from χ_i and χ_e. Zero value Dirichlet boundary conditions are used for $A_\parallel$ at the radial boundaries. The coefficient of the second term on the left hand side of (41) is not toroidally constant, so we again use either the multigrid or Naulin solvers. These solvers have been used successfully for filament simulations in slab geometry. However, as the vector variables are defined on the staggered grid, in tokamak geometry (41) must be solved on the x-z planes that intersect the X-points, see Fig. 2, which causes numerical problems. Several workarounds have been tried to enable electromagnetic runs in tokamak geometry, but so far without success.

3.2.3. Boundary conditions

The parallel boundary conditions, where the magnetic field intersects the wall at the divertor targets, are set by sheath physics, section 3.1.2. The scalar variables n, T_e, and ϕ should not have any parallel boundary condition, so to constrain them as little as possible, they are extrapolated to the sheath entrance with a quadratic polynomial extrapolation

$$f|_{\text{sheath}} = \frac{1}{8}\left(15 f_{i_{\text{end}}} - 10 f_{i_{\text{end}}\mp 1} + 3 f_{i_{\text{end}}\mp 2}\right), \tag{42}$$

where i_{end} is the y-index of the grid point adjacent to the sheath entrance, the upper signs apply to upper y-boundaries, and the lower signs to lower y-boundaries. Recall that the scalar variables are evolved on the cell centre grid, while the sheath entrance boundary is at a cell face location, so $f|_{\text{sheath}}$ is located at y-index $i_{\text{end}\pm 1/2}$. The extrapolated values of the scalar variables are used to evaluate the sheath entrance values of the parallel flows with (23) and (24), while the conductive parallel heat flux $q_{e\parallel}$ is calculated using the total parallel heat flux $Q_{e\parallel}$ evaluated similarly with (25) as

$$q_{e\parallel}\Big|_{\text{sheath}} = Q_{e\parallel}\Big|_{\text{sheath}} - \frac{5}{2} T_e\Big|_{\text{sheath}} V_{e\parallel}\Big|_{\text{sheath}} - \frac{1}{2} m_e \left(V_{e\parallel}\Big|_{\text{sheath}}\right)^3. \tag{43}$$

All variables are extrapolated into the boundary cells past the wall, again with a quadratic polynomial extrapolation

$$f_i = 3 f_{i\mp 1} - 3 f_{i\mp 2} + f_{i\mp 3}, \tag{44}$$

where i is the y-index of any grid point within the boundary. The parallel boundary conditions are applied to auxiliary variables containing the transformation to the field aligned grid of each quantity, as discussed in section 2.3.2. For some filament simulations, to reduce the computational expense, only half of the slab domain is simulated, dividing the domain in the parallel y-direction with boundary conditions at the lower-y end of the computational grid imposing reflection symmetry, i.e. Neumann boundary conditions for the scalar variables n, p_e, ϖ, ϕ and Dirichlet boundary conditions for the vector variables $V_{i\parallel}$, $V_{e\parallel}$, $A_\parallel$, $q_{e\parallel}$.

The radial boundaries of the grid are not physical. Their position should be chosen to be far enough away from the separatrix that the results of a simulation are insensitive to the exact position of the boundaries. Fluctuations that occur near the boundaries are therefore unimportant for the simulations, but have a tendency to lead to numerical instabilities. Therefore for turbulent simulations, a combination of radial boundary conditions is used that have been found from experience to mitigate these numerical instabilities by suppressing fluctuations (which are set to zero at the radial boundaries, and may be further damped by a buffer region with enhanced dissipation) but allowing the average values to float as freely as possible, to avoid creating artificial boundary layers. At the radial boundaries in SOL and PFR regions, $\log n$, $\log p_e$ and ϖ use a Neumann boundary condition on the toroidal-average part of the variable $\langle \log n\rangle_z$, etc. and a zero-value Dirichlet boundary condition on the 'fluctuating part' $\left(\log n - \langle \log n\rangle_z\right)$, etc. At the core boundary ϖ uses the same boundary condition as in the SOL and PFR, while the combined toroidal and poloidal average of $\log n$ or $\log p_e$, that is $\langle \log n\rangle_{yz}$ or $\langle \log p_e\rangle_{yz}$ use a Neumann boundary condition, while the 'fluctuating parts' which use Dirichlet boundary conditions are now $\left(\log n - \langle \log n\rangle_{yz}\right)$ and $\left(\log p_e - \langle \log p_e\rangle_{yz}\right)$. In filament simulations, Neumann boundary conditions are used at all radial boundaries for $\log n$, $\log p_e$, and ϖ. The radial boundary conditions for the electromagnetic potentials ϕ and $A_\parallel$ were described in section 3.2.2. In all simulations Neumann boundary conditions are used at all radial boundaries for the parallel velocities $V_{i\parallel}$ and $V_{e\parallel}$. Radial boundary conditions are applied, when necessary, to the intermediate variables storing interpolated, staggered versions of the variables in the same way as for the unstaggered ones and Neumann boundary conditions are used for $q_{e\parallel}$. The exceptions are the electromagnetic potentials. As the staggered versions of ϕ and $A_\parallel$ are calculated simply by interpolating, not by separate Laplace inversions (section 3.2.2), for simplicity 'free' boundary conditions are applied using quadratic extrapolation in the radial direction with the stencil (44).

Dissipative buffer regions can be created near the radial boundaries to further damp any fluctuations before they can interact with the boundary. These buffer regions include a number of grid points in the x-direction chosen for each simulation, typically 8 to 16. The perpendicular diffusion coefficients μ_n, $D_{V_{i\parallel}}$, $D_{V_{e\parallel}}$, $\kappa_{e\perp}$, and μ_ϖ are enhanced by a factor that increases linearly through each buffer region, up to a maximum enhancement factor at the radial boundary, which is again chosen for each simulation, and defaults to 10. It is also possible to enhance the parallel resistivity $0.51/\tau_{ei}$ by the same factor in the dissipative radial buffers; this enhancement is applied only in the ion and electron momentum equations (10) and (11), and not in the resistive heating term in the pressure equation (12) in order to avoid spurious heating in the buffers.

3.2.4. Time solver

As is standard for BOUT++ codes, STORM uses the method of lines. The spatial discretisation described above is used to implement the `STORM::rhs()` function which evaluates the time derivatives from the values of the evolving variables. This function is passed to the 'time solver' object, for which several implementation options are available in BOUT++ [22]. As the parallel thermal conduction of the electrons is a diffusive process which is faster than the typical turbulent dynamics, it would imply a very restrictive CFL condition for an explicit time stepping algorithm [19,20]. STORM therefore uses the CVODE library from the SUNDIALS suite, which uses a fully implicit, matrix free, variable-

order, variable-step multistep method, employing a Newton-Krylov iterative solver, to advance the solution in time [74]. Even without preconditioning, which has not yet been implemented in STORM,[4] the adaptive step size algorithm in CVODE provides a solution that is robust as the state of the system changes, for example during initial transients, which are often violent, before a settled turbulent state develops. The primary important settings are the relative and absolute tolerances that control the accuracy of the time advance.

As CVODE increases the internal time step, it sometimes happens that the time step gets too long, resulting in multiple iteration failures causing the time step to be decreased to a very small value before the simulation continues. This behaviour tends to happen in cycles and significantly increases the iteration count, and therefore the run time. It can be prevented by choosing an appropriate maximum timestep for the simulation, which should be as long as possible while avoiding the cycle just described.

3.3. Initialisation

Turbulence and seeded filament simulations have different requirements for initialisation, and are discussed in turn below.

In turbulence simulations, it is the long term, statistical steady state that is of interest, which should not be sensitive to the particular choice of initial conditions, but is instead determined by the sources and boundary conditions. The initial conditions are therefore arbitrary, but are chosen with the aim of allowing the steady state to be reached as quickly as possible. The best option is usually to restart from an existing turbulent state, even if it comes from a simulation with somewhat different parameters. When this is not possible, some analytical functions that satisfy the boundary conditions are used, with random noise added to seed turbulence. The initial values are defined in open field line regions as

$$\frac{n}{n_0} = 0.35 + 1.2\hat{y}(1-\hat{y}) \tag{45}$$

$$\frac{T_e}{T_0} = 0.5 + 0.23\hat{y}(1-\hat{y}) \tag{46}$$

$$\frac{V_{i\parallel}}{c_{s0}} = 5\hat{y}\left(\hat{y}^2-1\right) + \sqrt{\frac{T_e}{T_0}}(2\hat{y}-1), \tag{47}$$

in terms of a normalised coordinate $\hat{y}$ which varies proportional to y. In all open field line regions $\hat{y}$ goes from 0 at the lower-y target to 1 at the upper-y target. In SOL regions $\hat{y}$ simply varies linearly with y. In PFR regions $\hat{y}$ has two linear segments, defined so that $\hat{y} = 0.5$ at the y-position of the X-point. Similarly in the region between the two separatrices of a disconnected double null configuration $\hat{y}$ has two linear segments and $\hat{y} = 0.5$ is at the y-position of the secondary X-point. In closed field line regions the initial values are set to constant values with $V_{i\parallel} = 0$, while n and T_e are set to their maximum values in the open field line regions (45) and (46). In all regions, it is assumed that there is no current, so $V_{e\parallel} = V_{i\parallel}$ in the initial state and the electrostatic potential is set to

$$\phi = \frac{T_e}{e}\log\sqrt{\frac{m_i}{2\pi m_e}} \tag{48}$$

so that when evaluated at the sheath entrances, the boundary conditions there, (23) and (24), give zero parallel current. These initial profiles have no radial x-dependence, to minimise the radial gradients during the initial transient as turbulence starts to develop and to minimise the development of very low density far outside the separatrix in the phase before turbulence has developed enough to provide radial transport into this region. An example of these initial profiles for a disconnected double null equilibrium can be seen in figure 2 of [6].

Seeded filament simulations use a steady, time independent background plasma sustained by sources, see section 3.4. It is usual [33,35,64,76] to assume that the background plasma is constant in the radial and binormal directions, and varies only in the parallel direction. We therefore set up the initial background plasma by running a simulation in a reduced, one dimensional mode where perpendicular gradients are set to zero. In addition the parallel current is taken to vanish, so $V_{e\parallel}$ is set equal to $V_{i\parallel}$ and electron inertia is neglected, with Ohm's law (11) being used first to replace $\nabla_\parallel\phi$ in the ion velocity equation (10) and second to calculate ϕ by directly integrating from the boundary value that gives zero current at the sheath entrance from (23) and (24). In this one dimensional mode, ϕ is only calculated at time points when output is saved, because it is not directly needed to advance the simulation. The one dimensional simulation is run until a steady state is reached, and the final state is saved and used to initialise the background plasma for a three dimensional filament simulation. One or more filaments can be added on top of this background plasma. Their perpendicular cross sections are elliptical, described by Gaussian functions parameterised by a width δ_f, elongation ϵ_f and inclination α_f and centred on a specified radial position x_f and binormal position z_f. In the parallel direction, the filament has a finite length L_f, truncated by a hyperbolic tangent function with whose width is $\delta_{f\parallel}$. Combining these with an amplitude A, the density and temperature of each filament are given by a function of the form

$$\begin{aligned} &A\exp\left(-\frac{\left[\left(x-x_f\right)\cos\alpha_f + \left(z-z_f\right)\sin\alpha_f\right]^2}{\delta_f^2}\right. \\ &\qquad\left.-\frac{\left[-\left(x-x_0\right)\sin\alpha_f + \left(z-z_0\right)\cos\alpha_f\right]^2}{\left(\epsilon_f\delta_f\right)^2}\right) \\ &\times\frac{1}{2}\left(1-\tanh\left(\frac{y-\left(L_y+L_f\right)/2}{\delta_{f\parallel}/2}\right)\right) \\ &\times\frac{1}{2}\left(1-\tanh\left(\frac{\left(L_y+L_f\right)/2-y}{\delta_{f\parallel}/2}\right)\right), \end{aligned} \tag{49}$$

where L_y is the length of the simulation domain in the parallel y-direction and in the curved slab geometry, x, y, and z are orthogonal coordinates whose values are lengths in metres and $0 \le y \le L_y$. In simulations using a reflection symmetric lower-y boundary, the y-dependent part on the final two lines of (49) is replaced by $\left(1-\tanh\left(\left(y-L_f\right)/\delta_{f\parallel}\right)\right)/2$.

3.4. Source terms

The particle and energy sources S_n and S_E can be defined by arbitrary analytic functions of the coordinates, using expressions set in the input file. These expressions can become complicated when it is necessary to distinguish between different topological regions, e.g. core and PFR. For convenience more restricted functions are provided for the source terms usually used for turbulent simulations. There are sources to provide poloidally and toroidally uniform fuelling and heating in the core and the 'upstream' SOL (the part of the SOL adjacent to the core)

$$S_{n,\mathrm{core}} = S_{n,\mathrm{core}\,0}\exp\left(-\frac{\left(x-x_{n,\mathrm{core}}\right)^2}{w_{n,\mathrm{core}}^2}\right) \tag{50}$$

$$S_{E,\mathrm{core}} = S_{E,\mathrm{core}\,0}\exp\left(-\frac{\left(x-x_{E,\mathrm{core}}\right)^2}{w_{E,\mathrm{core}}^2}\right), \tag{51}$$

controlled by amplitudes $S_{n,\mathrm{core}\,0}$ and $S_{E,\mathrm{core}\,0}$, peak positions $x_{n,\mathrm{core}}$ and $x_{E,\mathrm{core}}$, and widths $w_{n,\mathrm{core}}$ and $w_{E,\mathrm{core}}$ that are set independently for the particle and energy sources. The energy source is usually positioned well

[4] Preconditioning the time advance has, however, been a topic of active research in the BOUT++ community [13,22,75].

inside the closed field line region, representing the outflow of heat from the core. The particle source is often positioned near the primary separatrix, roughly representing the ionisation source in attached divertor conditions where the low plasma density allows neutrals recycled from the target to travel upstream to the edge of the closed field line region. The amplitudes of both source terms are usually tuned to achieve the desired, time averaged density and temperature on the separatrix at the outboard midplane. There are also particle sources localised near the divertor targets

$$S_{n,\mathrm{target}} = \begin{cases} S_{n,\mathrm{target}\,0}\exp(-10\tilde{y}) & 0 \le \tilde{y} < 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{52}$$

where $\tilde{y}$ is a normalised coordinate proportional to y defined so that $\tilde{y} = 0$ at the targets and $\tilde{y} = 0.5$ at the y-position of the X-point nearest to the target. The divertor localised sources can be useful during the initial transient phase of turbulence simulations, to prevent numerical instabilities due to very low density near the targets while radial transport from the core is still at a low level. It can also be used as a very crude model of recycling in the divertors. To avoid very low temperatures in the initial transient phase, it is also possible to add a constant background energy source $S_{E,\mathrm{bg}} = n_0 T_0 \Omega_{i0}$. Note that sources added for numerical stability in the initial transient phase are removed before taking results in the saturated turbulent state.

Filament simulations use source terms to maintain a steady background plasma. The source terms are implemented using expressions in the input file so could be arbitrary, but studies with STORM have by convention used sources that are constant in the perpendicular directions to ensure that the background plasma has a stable, time independent steady state and vary exponentially in the parallel direction. The particle source is peaked near the targets, roughly representing recycling

$$S_n = S_{n0}\left[\exp\left(-\frac{y}{\Delta_{n,y}}\right) + \exp\left(-\frac{(L_y - y)}{\Delta_{n,y}}\right)\right], \tag{53}$$

and the energy source is peaked at the midplane, representing power exhausted from the core plasma

$$S_E = S_{E0}\exp\left(-\frac{\left|y - L_y/2\right|}{\Delta_{E,y}}\right). \tag{54}$$

The amplitudes S_{n0} and S_{E0} are tuned to give the desired density and temperature at the midplane and the decay lengths are set by $\Delta_{n,y}$ and $\Delta_{E,y}$. When a reflection symmetric lower-y boundary is used, the sources are instead $S_n = S_{n0}\exp\left(-\left(L_y - y\right)/\Delta_y\right)$ and $S_E = S_{E0}\exp\left(-y/\Delta_{E,y}\right)$.

3.5. Synthetic Langmuir probe diagnostics

A simple synthetic Langmuir probe signal can be generated by calculating the ion saturation current that a Langmuir probe, introduced into the plasma on a reciprocating probe, would measure, taking the plasma density and temperature as inputs [65]

$$j_{\mathrm{sat}} = \frac{1}{2}enc_s = \frac{1}{2}en\sqrt{\frac{T_e}{m_i + m_e}}, \tag{55}$$

where the factor of a half as compared to the Bohm sheath boundary condition (23) accounts for the density drop that would develop in the presheath between the unperturbed, 'upstream' plasma and the surface of a probe. Langmuir probes can take measurements at a high frequency ~ 1 MHz, and saving full outputs from the simulation at this rate would require a prohibitive amount of disk space. Therefore a generic interface was written [77] that allows some variables to be written out from selected grid points with a higher frequency than the full outputs. An arbitrary number of spatial locations where high frequency output will be written out can be selected in the input file.

4. Summary

Turbulence is critical to radial transport in the scrape-off layer of tokamak reactors, and is observed experimentally to produce large amplitude, coherent filaments that may propagate across the full width of the scrape-off layer. STORM has been developed to simulate scrape-off layer plasma turbulence, using a drift reduced, cold ion, collisional fluid model. STORM uses a flux surface aligned grid with field aligned parallel derivatives to deal efficiently with the strong anisotropy of a magnetised plasma, and evolves fluxes on a grid staggered in the parallel direction to avoid grid scale instabilities. The code is implemented using the BOUT++ framework, whose handling of staggered grids has been upgraded to support STORM's needs, including a new implementation for field aligned parallel derivatives.

CRediT authorship contribution statement

J.T. Omotani: Writing – original draft, Visualization, Software, Investigation, Conceptualization. **D. Dickinson:** Software. **B.D. Dudson:** Supervision, Software. **L. Easy:** Software, Investigation, Conceptualization. **D. Hoare:** Software. **P. Hill:** Software. **T. Nicholas:** Software, Investigation, Conceptualization. **J. Parker:** Software. **F. Riva:** Software, Investigation, Conceptualization. **N.R. Walkden:** Software, Investigation, Conceptualization. **Q. Xia:** Software, Investigation. **F. Militello:** Project administration, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Co-author is an editor for CPC - B. Dudson. The other authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work has been funded by the EPSRC Energy Programme [grant number EP/W006839/1]. To obtain further information on the data and models underlying this paper please contact PublicationsManager@ukaea.uk. This work was in part prepared by LLNL under Contract DE-AC52-07NA27344. This work used the ARCHER2 UK National Supercomputing Service (https://www.archer2.ac.uk) through the Plasma HEC consortium [grant number EP/R029148/1].

Appendix A. Syntax for thread based parallelism

The usual BOUT++ syntax uses C++ operator overloading to allow code that is compact and as close to the mathematical expression of the differential equations as possible. This is the style used in the example code listings in this paper, in the majority of the BOUT++ documentation [53], and in STORM. In this style, each C++ operator becomes an individual loop over the grid, with a single computation (for example addition of two array entries) inside the loop. The result is a large number of loops, each of which does a small amount of work. For thread-based parallelism, and also for the best vectorisation of the loops, this is not the optimal structure. There is an overhead cost to starting and stopping threads at the beginning and end of each loop, so to minimise the overhead the ideal structure would be a few loops each of which does a large amount of work. BOUT++ does support this style by providing a set of operators that act on a single grid point, so that an outer loop or loops can be written in the `PhysicsModel::rhs()` method. However, the 'outer loop' style is more cumbersome and requires more

care with the order of operations. For example, any operations that require Fourier transforms must be done outside the outer loops, because the Fourier transform couples the whole grid toroidally. There is therefore a trade off between maintainability, which is somewhat better for the operator-overloading style, and computational efficiency which is somewhat better for the outer loop style.

Appendix B. Normalised model equations

The normalisations defined in section 3.1.5 are

$$\begin{array}{lll} \hat{n} = n/n_0 & \hat{T}_e = T_e/T_0 & \hat{B} = B/B_0 \\ \hat{m}_e = m_e/m_i & \hat{\nabla} = \rho_{s0}\nabla & \partial/\partial\hat{t} = \Omega_{i0}^{-1}\partial/\partial t \\ \hat{\phi} = e\phi/T_0 & \hat{A}_\parallel = 2eA_\parallel/\beta_0 m_i c_{s0} & \hat{\varpi} = \varpi/en_0 \\ \hat{S}_n = S_n/n_0\Omega_{i0} & \hat{S}_E = S_E/n_0T_0\Omega_{i0} & \end{array} \tag{B.1}$$

using the reference density n_0, temperature T_0, magnetic field B_0 and derived parameters $\rho_{s0} = c_{s0}/\Omega_{i0}$, $c_{s0} = \sqrt{T_0/m_i}$, $\Omega_{i0} = eB_0/m_i$, and $\beta_0 = 2\mu_0 n_0 T_0/B_0^2$. Using these normalisations and applying the chain rule several times, the model equations (9)-(13) can be put into the form implemented in the code,

$$\begin{aligned} \frac{\partial \log\hat{n}}{\partial\hat{t}} =& -\left\{\hat{\phi}, \log\hat{n}\right\} - \hat{B}\hat{\nabla}_\parallel\left(\frac{\hat{V}_{e\parallel}}{\hat{B}}\right) - \hat{V}_{e\parallel}\hat{\nabla}_\parallel\log\hat{n} \\ & - \hat{C}(\hat{\phi}) + \frac{\hat{C}(\hat{p}_e)}{\hat{n}} + \hat{\mu}_n\frac{\hat{\nabla}_\perp^2\hat{n}}{\hat{n}} + \hat{\nabla}_\perp\left(\hat{\mu}_n\right)\cdot\hat{\nabla}_\perp\log\hat{n} \\ & + \frac{\hat{S}_n}{\hat{n}} \end{aligned}$$

(normalised electron continuity) (B.2)

$$\begin{aligned} \frac{\partial\hat{\chi}_i}{\partial\hat{t}} =& -\left\{\hat{\phi}, \hat{V}_{i\parallel}\right\} - \hat{V}_{i\parallel}\hat{\nabla}_\parallel\hat{V}_{i\parallel} - \hat{\nabla}_\parallel\hat{\phi} \\ & - \hat{m}_e\hat{\nu}_{\parallel 0}\frac{\hat{n}}{\hat{T}^{3/2}}\left(\hat{V}_{i\parallel} - \hat{V}_{e\parallel}\right) + 0.71\hat{\nabla}_\parallel\hat{T}_e \\ & + \hat{D}_{V_{i\parallel}}\hat{\nabla}_\perp^2\hat{V}_{i\parallel} - \frac{\hat{V}_{i\parallel}\hat{S}_n}{\hat{n}} \end{aligned}$$

(normalised ion velocity) (B.3)

$$\begin{aligned} \frac{\partial\hat{\chi}_e}{\partial\hat{t}} =& -\left\{\hat{\phi}, \hat{V}_{e\parallel}\right\} - \hat{V}_{e\parallel}\hat{\nabla}_\parallel\hat{V}_{e\parallel} + \frac{1}{\hat{m}_e}\hat{\nabla}_\parallel\hat{\phi} \\ & + \hat{\nu}_{\parallel 0}\frac{\hat{n}}{\hat{T}_e^{3/2}}\left(\hat{V}_{i\parallel} - \hat{V}_{e\parallel}\right) - \frac{1}{\hat{m}_e}\hat{T}_e\hat{\nabla}_\parallel\log\hat{n} \\ & - \frac{1.71}{\hat{m}_e}\hat{\nabla}_\parallel\hat{T}_e + \hat{D}_{V_{e\parallel}}\hat{\nabla}_\perp^2\hat{V}_{e\parallel} - \frac{\hat{V}_{e\parallel}\hat{S}_n}{\hat{n}} \end{aligned}$$

(normalised electron velocity) (B.4)

$$\begin{aligned} \frac{\partial\log\hat{p}_e}{\partial\hat{t}} =& -\left\{\hat{\phi}, \log\hat{p}_e\right\} - \hat{V}_{e\parallel}\hat{\nabla}_\parallel\log\hat{p}_e - \frac{2}{3\hat{p}_e}\hat{B}\hat{\nabla}_\parallel\left(\frac{\hat{q}_\parallel}{\hat{B}}\right) \\ & - \frac{2}{3}0.71\left(\hat{V}_{i\parallel} - \hat{V}_{e\parallel}\right)\hat{\nabla}_\parallel\log\hat{T}_e - \frac{5}{3}\hat{B}\hat{\nabla}_\parallel\left(\frac{\hat{V}_{e\parallel}}{\hat{B}}\right) \\ & + \frac{2\hat{m}_e}{3}\hat{\nu}_{\parallel 0}\frac{\hat{n}}{\hat{T}_e^{5/2}}\left(\hat{V}_{i\parallel} - \hat{V}_{e\parallel}\right)^2 \\ & + \frac{5}{3}\left(-\hat{C}(\hat{\phi}) + \frac{\hat{C}(\hat{p}_e)}{\hat{n}} + \hat{C}(\hat{T}_e)\right) - \frac{\hat{m}_e\hat{V}_{e\parallel}^2}{3\hat{p}_e}\hat{C}(\hat{p}_e) \\ & + \frac{2}{3\hat{p}_e}\left(\hat{\kappa}_\perp\hat{\nabla}_\perp^2\hat{T}_e + \hat{\nabla}_\perp\left(\hat{\kappa}_\perp\right)\cdot\hat{\nabla}_\perp\hat{T}_e\right) \\ & + \frac{2}{3\hat{p}_e}\hat{S}_E + \frac{\hat{m}_e\hat{V}_{e\parallel}^2}{3\hat{p}_e}\hat{S}_n \end{aligned}$$

(normalised electron pressure) (B.5)

$$\begin{aligned} \frac{\partial\hat{\varpi}}{\partial\hat{t}} =& -\left\{\hat{\phi}, \hat{\varpi}\right\} - \hat{V}_{i\parallel}\hat{\nabla}_\parallel\hat{\varpi} \\ & + \hat{n}\hat{B}\hat{\nabla}_\parallel\left(\frac{\hat{V}_{i\parallel} - \hat{V}_{e\parallel}}{\hat{B}}\right) + \hat{n}\left(\hat{V}_{i\parallel} - \hat{V}_{e\parallel}\right)\hat{\nabla}_\parallel\log\hat{n} \\ & + \hat{C}(\hat{p}_e) + \hat{\mu}_\varpi\hat{\nabla}_\perp^2\hat{\varpi} + \hat{\nabla}_\perp\left(\hat{\mu}_\varpi\right)\cdot\hat{\nabla}_\perp\hat{\varpi}. \end{aligned}$$

(normalised vorticity) (B.6)

The bracket operator is $\left\{\hat{f}, \hat{h}\right\} = \hat{B}^{-1}\boldsymbol{b}\cdot\hat{\nabla}\hat{f}\times\hat{\nabla}\hat{h}$. The normalised electron pressure is $\hat{p}_e = \hat{n}\hat{T}_e$. The composite variables $\hat{\chi}_i = \hat{V}_{i\parallel} + \frac{\beta_0}{2}\hat{A}_\parallel$ and $\hat{\chi}_e = \hat{V}_{e\parallel} - \frac{\beta_0}{2\hat{m}_e}\hat{A}_\parallel$ are used in the momentum equations and in electromagnetic mode Ampère's law is used in the form

$$\hat{n}\left(\hat{V}_{i\parallel} - \hat{V}_{e\parallel}\right) = -\hat{\nabla}_\perp^2\hat{A}_\parallel. \tag{B.7}$$

The normalised, generalised vorticity is

$$\hat{\varpi} = \hat{\nabla}\cdot\left(\frac{1}{\hat{B}^2}\hat{\nabla}_\perp\hat{\phi}\right) = \frac{1}{\hat{B}^2}\hat{\nabla}_\perp^2\hat{\phi} + \hat{\nabla}_\perp\left(\frac{1}{\hat{B}^2}\right)\cdot\hat{\nabla}_\perp\hat{\phi} \tag{B.8}$$

and the conductive parallel heat flux is

$$\hat{q}_\parallel = -\hat{\kappa}_0\hat{T}_e^{5/2}\hat{\nabla}_\parallel\hat{T}_e - 0.71\hat{n}\hat{T}_e\left(\hat{V}_{i\parallel} - \hat{V}_{e\parallel}\right). \tag{B.9}$$

The default values of the constant prefactors in the parallel dissipation coefficients are

$$\hat{\nu}_{\parallel 0} = 0.51\frac{1}{\tau_{ei0}\Omega_{i0}} \tag{B.10}$$

$$\hat{\kappa}_0 = 3.16\frac{T_0\tau_{ei0}}{m_e\rho_{s0}^2\Omega_{i0}}, \tag{B.11}$$

where a collision time evaluated at the reference parameters is $\tau_{ab0} = 12\pi^{3/2}\epsilon_0^2 m_a^{1/2}T_0^{3/2}/2^{1/2}n_0e^4\ln\Lambda$. The default perpendicular dissipation coefficients are

$$\hat{\mu}_n = \left(1 + 1.3q^2\right)2\frac{1}{\hat{m}_e\tau_{ei0}\Omega_{i0}}\frac{\hat{n}}{\hat{B}^2\sqrt{\hat{T}_e}} \tag{B.12}$$

$$\hat{\kappa}_\perp = \left(1 + 1.6q^2\right)4.66\frac{1}{\hat{m}_e\tau_{ei0}\Omega_{i0}}\frac{\hat{n}^2}{\hat{B}^2\sqrt{\hat{T}_e}} \tag{B.13}$$

$$\hat{\mu}_\varpi = \left(1 + 1.6q^2\right)\frac{3}{4}\frac{1}{\tau_{ii0}\Omega_{i0}}\frac{\hat{n}}{\hat{B}^2\sqrt{\hat{T}_e}}, \tag{B.14}$$

but these coefficients are often set to constant values purely to give numerical dissipation that prevents fluctuations reaching the grid scale, see section 3.1.3. $\hat{D}_{V_{i\parallel}}$ and $\hat{D}_{V_{e\parallel}}$ are only introduced for numerical reasons, and are given constant numerical values.

B.1. 'Boussinesq approximation' variations

If the original STORM Boussinesq approximation is selected, the normalised form of the modified, generalised vorticity (19) is

$$\hat{\varpi}_{\mathrm{mod}} = \hat{\nabla}\cdot\left(\frac{1}{\hat{B}^2}\hat{\nabla}_\perp\hat{\phi}\right) = \frac{1}{\hat{B}^2}\hat{\nabla}_\perp^2\hat{\phi} + \hat{\nabla}_\perp\left(\frac{1}{\hat{B}^2}\right)\cdot\hat{\nabla}_\perp\hat{\phi} \tag{B.15}$$

and the corresponding vorticity equation (20) becomes

$$\begin{aligned} \frac{\partial\hat{\varpi}_{\mathrm{mod}}}{\partial\hat{t}} =& -\left\{\hat{\phi}, \hat{\varpi}_{\mathrm{mod}}\right\} - \hat{V}_{i\parallel}\hat{\nabla}_\parallel\hat{\varpi}_{\mathrm{mod}} \\ & + \hat{B}\hat{\nabla}_\parallel\left(\frac{\hat{V}_{i\parallel} - \hat{V}_{e\parallel}}{\hat{B}}\right) + \left(\hat{V}_{i\parallel} - \hat{V}_{e\parallel}\right)\hat{\nabla}_\parallel\log\hat{n} \\ & + \frac{\hat{C}(\hat{p}_e)}{\hat{n}} + \hat{\mu}_\varpi\hat{\nabla}_\perp^2\hat{\varpi}_{\mathrm{mod}} + \hat{\nabla}_\perp\left(\hat{\mu}_\varpi\right)\cdot\hat{\nabla}_\perp\hat{\varpi}_{\mathrm{mod}}. \end{aligned} \tag{B.16}$$

Without any Boussinesq approximation, the normalised form of the generalised vorticity (21) is

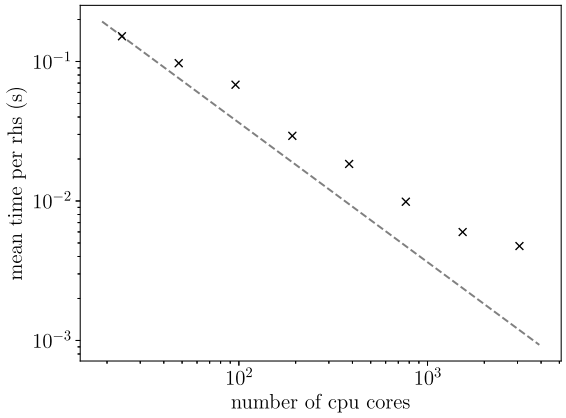


Fig. D.7. Strong scaling performance of STORM on the ARCHER2 cluster, for a simulation run with $256 \times 96 \times 64$ grid points.

$$\hat{\varpi}_{\mathrm{full}} = \hat{\nabla} \cdot \left(\frac{\hat{n}}{\hat{B}^2} \hat{\nabla}_\perp \hat{\phi} \right) = \frac{\hat{n}}{\hat{B}^2} \hat{\nabla}_\perp^2 \hat{\phi} + \hat{\nabla}_\perp \left(\frac{\hat{n}}{\hat{B}^2} \right) \cdot \hat{\nabla}_\perp \hat{\phi} \tag{B.17}$$

and the vorticity equation (22) becomes

$$\begin{aligned} \frac{\partial \hat{\varpi}_{\mathrm{full}}}{\partial \hat{t}} = &- \left\{ \hat{\phi}, \hat{\varpi}_{\mathrm{full}} \right\} - \left\{ \frac{1}{2} \left| \frac{\boldsymbol{b} \times \hat{\nabla} \hat{\phi}}{\hat{B}} \right|^2, \hat{n} \right\} - \hat{V}_{i\parallel} \hat{\nabla}_\parallel \hat{\varpi}_{\mathrm{full}} \\ &+ \hat{n} \hat{B} \hat{\nabla}_\parallel \left(\frac{\hat{V}_{i\parallel} - \hat{V}_{e\parallel}}{\hat{B}} \right) + \hat{n} \left(\hat{V}_{i\parallel} - \hat{V}_{e\parallel} \right) \hat{\nabla}_\parallel \log \hat{n} \\ &+ \hat{C}(\hat{p}_e) + \hat{\mu}_\varpi \hat{\nabla}_\perp^2 \hat{\varpi}_{\mathrm{full}} + \hat{\nabla}_\perp \left(\hat{\mu}_\varpi \right) \cdot \hat{\nabla}_\perp \hat{\varpi}_{\mathrm{full}}. \end{aligned} \tag{B.18}$$

Appendix C. Curvature for slab geometry

For simulations in curved slab geometry, STORM uses the curvature as calculated for a purely toroidal magnetic field, like that produced by an infinite straight wire. A cylindrical coordinate system is given by defining the major radius R as the distance away from the wire, Z as the distance along the wire, and choosing the direction of the magnetic field so that $\{x, y, z\}$ is a right handed coordinate system with unit vectors along the coordinate axes given by $\hat{\boldsymbol{x}} = \nabla R$, $\hat{\boldsymbol{y}} = \hat{\boldsymbol{z}} \times \hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}$ and $\hat{\boldsymbol{z}} = \nabla Z$. The magnetic field strength is proportional to R^{-1}, so $\boldsymbol{B}$ can be written as $\boldsymbol{B} = B\hat{\boldsymbol{y}}$ with $B = B_0 R_\mathrm{c}/R$ where R_c is the radius of curvature at the points where $B = B_0$. Noting that there is no current outside the wire, $\nabla \times \boldsymbol{B} = 0$ for $R > 0$. Therefore the curvature (15) for this magnetic field is

$$\begin{aligned} \nabla \times \left(\frac{\boldsymbol{b}}{B} \right) &= \nabla \times \left(\frac{\boldsymbol{B}}{B^2} \right) \\ &= -\boldsymbol{B} \times \nabla \left(\frac{1}{B^2} \right) \\ &= -B\hat{\boldsymbol{y}} \times \frac{2R}{B_0^2 R_\mathrm{c}^2} \hat{\boldsymbol{x}} \\ &\approx \frac{2}{B_0 R_\mathrm{c}} \hat{\boldsymbol{z}}, \end{aligned} \tag{C.1}$$

where on the last line we evaluate the expression at $R = R_\mathrm{c}$ under the assumption that the width of the simulation box is much smaller than R_c.

Appendix D. Strong scaling performance

An example strong scaling study of the performance of STORM has been conducted on the ARCHER2 HPC cluster [78] (see Fig. D.7). Each node on ARCHER2 has two AMD EPYC 7742 64-core 2.25 GHz processors and 256 GB of RAM, and nodes are connected by a HPE Slingshot interconnect. The test used a simulation of MAST on a grid with 256 points in the radial x-direction, 96 in the parallel y-direction and 64 in the toroidal z-direction. The plot shows the mean time per `STORM::rhs()` evaluation over a short test simulation, restarted from a saturated turbulent state. Simulating 1 ms of plasma turbulence takes of the order of 10^8 `STORM::rhs()` evaluations, corresponding to around two weeks on 768 cores, although this will vary depending on the grid, plasma conditions, etc.

Appendix E. Post processing

Post processing tools for BOUT++ are provided by the xBOUT Python package [79], which provides a function to load BOUT++ output into an xarray [80,81] `Dataset` object, with extra BOUT++ specific functionality provided by the `BoutDataset` accessor.

We have written an extension of xBOUT called xSTORM to provide further functionality specific to STORM, which is included with the STORM code. STORM output can be converted to SI units from the internal, dimensionless representation. Particle fluxes can be calculated consistent with the terms as implemented in STORM. Values of variables at the position of the sheath entrance can be re-calculated with the same algorithm as STORM uses. Various convenience methods are included for statistical analysis of turbulence and for frequently used operations in analysing seeded filament simulations, such as calculating the position and velocity of the filament centre of mass.

xBOUT and xSTORM provide high-level functions for data analysis and visualisation. xarray supports a method chaining syntax which allows operations on a data set, or a variable, to be combined in a single line while maintaining an easy to read left-to-right order of method application. Taken together, these two features support a workflow where outputs can be generated with just a few lines of code in a Python script, interactive session or Jupyter notebook, which has proven to be beneficial for productivity, interactivity and code reuse.

Appendix F. Provenance tracking

Provenance tracking and reproducibility of simulations is important for the reliability of the research process. For example, if a published result is contradicted by a new study it is important to be able to find the reason for the difference: Is it some new physics? Better resolution achieved on newer, more powerful computers? A bug in the old code, or in the new code? While these questions may be difficult to answer in general, they become almost impossible if it is not known what software was used to produce the result.

Version control provides a first step. Both BOUT++ and STORM are version controlled using git and both save the hash which identifies the commit used to build the executable into the output files for each simulation. STORM in addition saves the difference in the STORM repo between the compiled code and the last commit (the output of `git diff`).

The versions of compilers and external libraries used to build the code may also be important. To try to capture this information, STORM saves the contents of the `CMakeCache.txt` file produced by CMake. The module system is commonly used to manage libraries on HPC clusters; when it is present STORM also saves the active modules when it was compiled (the output of `module list`). Complete configuration for the HPC systems usually used by our group is stored in version controlled bash scripts; the git hash and git diff for the repos containing these configuration scripts are recorded.

The inputs used for a simulation are also critical. It is good practice to archive the input file along with the output. However, SOL turbulence simulations take many days, or even several weeks, to run and so the output is produced over many restarts of the code. Output is usually stored in separate files for each restart in order to avoid excessively large files which can cause problems for storage systems and increase the risk of data corruption. In order to verify that a set of files belongs to a consecutive sequence of restarts, BOUT++ assigns a standard universally unique identifier (UUID) [82] to each run. When restarting, it

also records the UUID of the run that produced the snapshot used to restart from. STORM records in addition the contents of the input file used for the run and also the input files used for all the previous restarts into the output binary (NetCDF) files. This feature provides a fallback in case output files from some of the restarts leading up to a certain run are misplaced. It is also a useful record as, especially in the early stages of a simulation where there is usually a violent initial transient, it is sometimes necessary to change the settings, e.g. for numerical dissipation, between restarts. The grid file that represents the magnetic equilibrium is the other important input. The grid generator [58] assigns a UUID to each grid file produced, and the UUID from the grid file is recorded by BOUT++ in simulation output.

The provenance tracking information described above is all written into the same binary (NetCDF) files as the simulation output, minimising the likelihood that it will be separated from the output. The features have been introduced incrementally and not all are present in older versions of STORM. The information is not perfect, for example if options are passed on the command line they are recorded only in log files, not binary output, or on unsupported systems configuration information such as library dependencies might well not be recorded. Nevertheless, for our usual workflows this information provides a high degree of reproducibility.

Appendix G. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cpc.2025.109893.

Data availability

No data was used for the research described in the article.

References

[1] H. Zohm, C. Angioni, E. Fable, G. Federici, G. Gantenbein, T. Hartmann, K. Lackner, E. Poli, L. Porte, O. Sauter, G. Tardini, D. Ward, M. Wischmeier, On the physics guidelines for a tokamak DEMO, Nucl. Fusion 53 (7) (2013) 073019, https://doi.org/10.1088/0029-5515/53/7/073019.

[2] S.I. Braginskii, Transport processes in a plasma, Rev. Plasma Phys. 1 (1965) 205, authorized translation from the Russian by Herbert Lashinsky, University of Maryland.

[3] J.F. Drake, T.M. Antonsen, Nonlinear reduced fluid equations for toroidal plasmas, Phys. Fluids 27 (4) (1984) 898–908, https://aip.scitation.org/doi/pdf/10.1063/1.864680, https://doi.org/10.1063/1.864680, https://aip.scitation.org/doi/abs/10.1063/1.864680.

[4] A.N. Simakov, P.J. Catto, Drift-ordered fluid equations for field-aligned modes in low-β collisional plasma with equilibrium pressure pedestals, Phys. Plasmas 10 (12) (2003) 4744–4757, https://doi.org/10.1063/1.1623492.

[5] F. Wagner, G. Becker, K. Behringer, D. Campbell, A. Eberhagen, W. Engelhardt, G. Fussmann, O. Gehre, J. Gernhardt, G.v. Gierke, G. Haas, M. Huang, F. Karger, M. Keilhacker, O. Klüber, M. Kornherr, K. Lackner, G. Lisitano, G.G. Lister, H.M. Mayer, D. Meisel, E.R. Müller, H. Murmann, H. Niedermeyer, W. Poschenrieder, H. Rapp, H. Röhr, F. Schneider, G. Siller, E. Speth, A. Stäbler, K.H. Steuer, G. Venus, O. Vollmer, Z. Yü, Regime of improved confinement and high beta in neutral-beam-heated divertor discharges of the asdex tokamak, Phys. Rev. Lett. 49 (1982) 1408–1412, https://doi.org/10.1103/PhysRevLett.49.1408, https://link.aps.org/doi/10.1103/PhysRevLett.49.1408.

[6] F. Riva, F. Militello, S. Elmore, J.T. Omotani, B. Dudson, N.R. Walkden, The MAST Team, Three-dimensional plasma edge turbulence simulations of the mega ampere spherical tokamak and comparison with experimental measurements, Plasma Phys. Control. Fusion 61 (9) (2019) 095013, https://doi.org/10.1088/1361-6587/ab3561.

[7] W. Zholobenko, T. Body, P. Manz, A. Stegmeir, B. Zhu, M. Griener, G.D. Conway, D. Coster, F. Jenko, Electric field and turbulence in global Braginskii simulations across the ASDEX upgrade edge and scrape-off layer, Plasma Phys. Control. Fusion 63 (3) (2021) 034001, https://doi.org/10.1088/1361-6587/abd97e.

[8] M. Wiesenberger, L. Einkemmer, M. Held, A. Gutierrez-Milla, X. Sáez, R. Iakymchuk, Reproducibility, accuracy and performance of the Feltor code and library on parallel computer architectures, Comput. Phys. Commun. 238 (2019) 145–156, https://doi.org/10.1016/j.cpc.2018.12.006, https://www.sciencedirect.com/science/article/pii/S0010465518304223.

[9] F. Halpern, P. Ricci, S. Jolliet, J. Loizu, J. Morales, A. Mosetto, F. Musil, F. Riva, T. Tran, C. Wersal, The GBS code for tokamak scrape-off layer simulations, J. Comput. Phys. 315 (2016) 388–408, https://doi.org/10.1016/j.jcp.2016.03.040, https://www.sciencedirect.com/science/article/pii/S0021999116001923.

[10] B. Zhu, M. Francisquez, B.N. Rogers, GDB: a global 3D two-fluid model of plasma turbulence and transport in the tokamak edge, Comput. Phys. Commun. 232 (2018) 46–58, https://doi.org/10.1016/j.cpc.2018.06.002, https://www.sciencedirect.com/science/article/pii/S001046551830208X.

[11] A. Stegmeir, D. Coster, A. Ross, O. Maj, K. Lackner, E. Poli, GRILLIX: a 3D turbulence code based on the flux-coordinate independent approach, Plasma Phys. Control. Fusion 60 (3) (2018) 035005, https://doi.org/10.1088/1361-6587/aaa373.

[12] B.D. Dudson, J. Leddy, Hermes: global plasma edge fluid turbulence simulations, Plasma Phys. Control. Fusion 59 (5) (2017) 054010, https://doi.org/10.1088/1361-6587/aa63d2.

[13] B. Dudson, M. Kryjak, H. Muhammed, P. Hill, J. Omotani, Hermes-3: multi-component plasma simulations with bout++, Comput. Phys. Commun. 296 (2024) 108991, https://doi.org/10.1016/j.cpc.2023.108991, https://www.sciencedirect.com/science/article/pii/S0010465523003363.

[14] H. Bufferand, P. Tamain, S. Baschetti, J. Bucalossi, G. Ciraolo, N. Fedorczak, P. Ghendrih, F. Nespoli, F. Schwander, E. Serre, Y. Marandet, Three-dimensional modelling of edge multi-component plasma taking into account realistic wall geometry, Nucl. Mater. Energy 18 (2019) 82–86, https://doi.org/10.1016/j.nme.2018.11.025, https://www.sciencedirect.com/science/article/pii/S2352179118302035.

[15] R. Hager, S. Ku, A.Y. Sharma, C.S. Chang, R.M. Churchill, A. Scheinberg, Electromagnetic total-f algorithm for gyrokinetic particle-in-cell simulations of boundary plasma in xgc, Phys. Plasmas 29 (11) (2022) 112308, arXiv:https://pubs.aip.org/aip/pop/article-pdf/doi/10.1063/5.0097855/16627866/112308_1_online.pdf, https://doi.org/10.1063/5.0097855.

[16] M. Dorf, M. Dorr, Continuum gyrokinetic simulations of edge plasmas in single-null geometries, Phys. Plasmas 28 (3) (2021) 032508, arXiv:https://pubs.aip.org/aip/pop/article-pdf/doi/10.1063/5.0039169/12361228/032508_1_online.pdf, https://doi.org/10.1063/5.0039169.

[17] A.H. Hakim, N.R. Mandell, T.N. Bernard, M. Francisquez, G.W. Hammett, E.L. Shi, Continuum electromagnetic gyrokinetic simulations of turbulence in the tokamak scrape-off layer and laboratory devices, Phys. Plasmas 27 (4) (2020) 042304, arXiv:https://pubs.aip.org/aip/pop/article-pdf/doi/10.1063/1.5141157/15923398/042304_1_online.pdf, https://doi.org/10.1063/1.5141157.

[18] D. Michels, A. Stegmeir, P. Ulbl, D. Jarema, F. Jenko, Gene-x: a full-f gyrokinetic turbulence code based on the flux-coordinate independent approach, Comput. Phys. Commun. 264 (2021) 107986, https://doi.org/10.1016/j.cpc.2021.107986, https://www.sciencedirect.com/science/article/pii/S0010465521000989.

[19] H. Lewy, K. Friedrichs, R. Courant, Über die partiellen differenzengleichungen der mathematischen physik, Math. Ann. 100 (1928) 32–74, https://doi.org/10.1007/BF01448839, http://eudml.org/doc/159283.

[20] R. Courant, K. Friedrichs, H. Lewy, On the partial difference equations of mathematical physics, IBM J. Res. Dev. 11 (2) (1967) 215–234, https://doi.org/10.1147/rd.112.0215.

[21] B. Dudson, M. Umansky, X. Xu, P. Snyder, H. Wilson, BOUT++: a framework for parallel plasma fluid simulations, Comput. Phys. Commun. 180 (9) (2009) 1467–1480, https://doi.org/10.1016/j.cpc.2009.03.008, https://www.sciencedirect.com/science/article/pii/S0010465509001040.

[22] B.D. Dudson, A. Allen, G. Breyiannis, E. Brugger, J. Buchanan, L. Easy, S. Farley, I. Joseph, M. Kim, A.D. McGann, et al., BOUT++: recent and current developments, J. Plasma Phys. 81 (1) (2015) 365810104, https://doi.org/10.1017/S0022377814000816.

[23] B.D. Dudson, J. Madsen, J. Omotani, P. Hill, L. Easy, M. Løiten, Verification of BOUT++ by the method of manufactured solutions, Phys. Plasmas 23 (6) (2016) 062303, https://doi.org/10.1063/1.4953429.

[24] F. Hariri, M. Ottaviani, A flux-coordinate independent field-aligned approach to plasma turbulence simulations, Comput. Phys. Commun. 184 (11) (2013) 2419–2429, https://doi.org/10.1016/j.cpc.2013.06.005, https://www.sciencedirect.com/science/article/pii/S0010465513001999.

[25] B.W. Shanahan, P. Hill, B.D. Dudson, Towards nonaxisymmetry; initial results using the flux coordinate independent method in BOUT++, J. Phys. Conf. Ser. 775 (2016) 012012, https://doi.org/10.1088/1742-6596/775/1/012012.

[26] B. Shanahan, B. Dudson, P. Hill, Fluid simulations of plasma filaments in stellarator geometries with BSTING, Plasma Phys. Control. Fusion 61 (2) (2018) 025007, https://doi.org/10.1088/1361-6587/aaed7d.

[27] D.A. D'Ippolito, J.R. Myra, S.J. Zweben, Convective transport by intermittent blob-filaments: comparison of theory and experiment, Phys. Plasmas 18 (6) (2011) 060501, https://doi.org/10.1063/1.3594609.

[28] R.J. Maqueda, G.A. Wurden, S. Zweben, L. Roquemore, H. Kugel, D. Johnson, S. Kaye, S. Sabbagh, R. Maingi, Edge turbulence measurements in NSTX by gas puff imaging, Rev. Sci. Instrum. 72 (1) (2001) 931–934, https://doi.org/10.1063/1.1321009.

[29] S. Zweben, R. Maqueda, D. Stotler, A. Keesee, J. Boedo, C. Bush, S. Kaye, B. LeBlanc, J. Lowrance, V. Mastrocola, R. Maingi, N. Nishino, G. Renda, D. Swain, J. Wilgen, The NSTX Team, High-speed imaging of edge turbulence in NSTX, Nucl. Fusion 44 (1) (2003) 134–153, https://doi.org/10.1088/0029-5515/44/1/016.

[30] N.B. Ayed, A. Kirk, B. Dudson, S. Tallents, R.G.L. Vann, H.R. Wilson, The MAST Team, Inter-ELM filaments and turbulent transport in the mega-amp spherical tokamak, Plasma Phys. Control. Fusion 51 (3) (2009) 035016, https://doi.org/10.1088/0741-3335/51/3/035016.

[31] J.R. Harrison, G.M. Fishpool, A.J. Thornton, N.R. Walkden, The appearance and propagation of filaments in the private flux region in mega amp spherical tokamak, Phys. Plasmas 22 (9) (2015) 092508, https://doi.org/10.1063/1.4929924.

[32] C. Killer, B. Shanahan, O. Grulke, M. Endler, K. Hammond, L. Rudischhauser, Plasma filaments in the scrape-off layer of Wendelstein 7-X, Plasma Phys. Control. Fusion 62 (8) (2020) 085003, https://doi.org/10.1088/1361-6587/ab9313.

[33] L. Easy, F. Militello, J. Omotani, B. Dudson, E. Havlíčková, P. Tamain, V. Naulin, A.H. Nielsen, Three dimensional simulations of plasma filaments in the scrape off layer: a comparison with models of reduced dimensionality, Phys. Plasmas 21 (12) (2014) 122515, https://doi.org/10.1063/1.4904207.

[34] L. Easy, F. Militello, J. Omotani, N.R. Walkden, B. Dudson, Investigation of the effect of resistivity on scrape off layer filaments using three-dimensional simulations, Phys. Plasmas 23 (1) (2016) 012512, https://doi.org/10.1063/1.4940330.

[35] N.R. Walkden, B.D. Dudson, G. Fishpool, Characterization of 3d filament dynamics in a MAST SOL flux tube geometry, Plasma Phys. Control. Fusion 55 (10) (2013) 105005, https://doi.org/10.1088/0741-3335/55/10/105005.

[36] N. Walkden, B. Dudson, L. Easy, G. Fishpool, J. Omotani, Numerical investigation of isolated filament motion in a realistic tokamak geometry, Nucl. Fusion 55 (11) (2015) 113022, https://doi.org/10.1088/0029-5515/55/11/113022.

[37] N.R. Walkden, L. Easy, F. Militello, J.T. Omotani, Dynamics of 3d isolated thermal filaments, Plasma Phys. Control. Fusion 58 (11) (2016) 115010, https://doi.org/10.1088/0741-3335/58/11/115010.

[38] F. Militello, N.R. Walkden, T. Farley, W.A. Gracias, J. Olsen, F. Riva, L. Easy, N. Fedorczak, I. Lupelli, J. Madsen, A.H. Nielsen, P. Ricci, P. Tamain, J. Young, Multi-code analysis of scrape-off layer filament dynamics in MAST, Plasma Phys. Control. Fusion 58 (10) (2016) 105002, https://doi.org/10.1088/0741-3335/58/10/105002.

[39] F. Militello, B. Dudson, L. Easy, A. Kirk, P. Naylor, On the interaction of scrape off layer filaments, Plasma Phys. Control. Fusion 59 (12) (2017) 125013, https://doi.org/10.1088/1361-6587/aa9252.

[40] D. Schwörer, N. Walkden, H. Leggate, B. Dudson, F. Militello, T. Downes, M. Turner, Influence of plasma background including neutrals on scrape-off layer filaments using 3d simulations, in: Proceedings of the 22nd International Conference on Plasma Surface Interactions 2016, 22nd PSI, Nucl. Mater. Energy 12 (2017) 825–830, https://doi.org/10.1016/j.nme.2017.02.016, https://www.sciencedirect.com/science/article/pii/S2352179116301648.

[41] D. Schwörer, N.R. Walkden, H. Leggate, B.D. Dudson, F. Militello, T. Downes, M.M. Turner, Influence of plasma background on 3d scrape-off layer filaments, Plasma Phys. Control. Fusion 61 (2) (2018) 025008, https://doi.org/10.1088/1361-6587/aae8fe.

[42] D. Hoare, F. Militello, J.T. Omotani, F. Riva, S. Newton, T. Nicholas, D. Ryan, N.R. Walkden, Dynamics of scrape-off layer filaments in high β plasmas, Plasma Phys. Control. Fusion 61 (10) (2019) 105013, https://doi.org/10.1088/1361-6587/ab34f8.

[43] S.I. Krasheninnikov, D.A. D'Ippolito, J.R. Myra, Recent theoretical progress in understanding coherent structures in edge and sol turbulence, J. Plasma Phys. 74 (5) (2008) 679–717, https://doi.org/10.1017/S0022377807006940.

[44] J.T. Omotani, F. Militello, L. Easy, N.R. Walkden, The effects of shape and amplitude on the velocity of scrape-off layer filaments, Plasma Phys. Control. Fusion 58 (1) (2015) 014030, https://doi.org/10.1088/0741-3335/58/1/014030.

[45] T.E.G. Nicholas, J. Omotani, F. Riva, F. Militello, B. Dudson, Comparing two- and three-dimensional models of scrape-off layer turbulent transport, Plasma Phys. Control. Fusion 64 (9) (2022) 095001, https://doi.org/10.1088/1361-6587/ac7b48.

[46] G. Decristoforo, F. Militello, T. Nicholas, J. Omotani, C. Marsden, N. Walkden, O.E. Garcia, Blob interactions in 2d scrape-off layer simulations, Phys. Plasmas 27 (12) (2020) 122301, https://doi.org/10.1063/5.0021314.

[47] G. Decristoforo, A. Theodorsen, J. Omotani, T. Nicholas, O.E. Garcia, Numerical turbulence simulations of intermittent fluctuations in the scrape-off layer of magnetized plasmas, Phys. Plasmas 28 (7) (2021) 072301, https://doi.org/10.1063/5.0047566.

[48] L. Easy, 3d simulations of scrape-off layer filaments, Ph.D. thesis, University of York, 2016, https://etheses.whiterose.ac.uk/15850/.

[49] T. Nicholas, Reduced simulations of scrape-off-layer turbulence, Ph.D. thesis, University of York, 2021, https://etheses.whiterose.ac.uk/29962/.

[50] N. Walkden, F. Riva, B. Dudson, C. Ham, F. Militello, D. Moulton, T. Nicholas, J. Omotani, 3d simulations of turbulent mixing in a simplified slab-divertor geometry, Nucl. Mater. Energy 18 (2019) 111–117, https://doi.org/10.1016/j.nme.2018.12.005, https://www.sciencedirect.com/science/article/pii/S2352179118301492.

[51] N. Walkden, F. Riva, J. Harrison, F. Militello, T. Farley, J. Omotani, B. Lipschultz, The physics of turbulence localised to the tokamak divertor volume, Commun. Phys. 5 (1) (2022) 139, https://doi.org/10.1038/s42005-022-00906-2.

[52] Q. Xia, D. Moulton, J. Omotani, F. Militello, The effect of divertor particle sources on scrape-off-layer turbulence, Plasma Phys. Control. Fusion 66 (6) (2024) 065022, https://doi.org/10.1088/1361-6587/ad441c.

[53] The BOUT++ Team, Bout++'s documentation, https://bout-dev.readthedocs.io/, 2025.

[54] B. Dudson, P. Hill, D. Dickinson, J. Parker, A. Allen, D. Bold, G. Breyiannis, J. Brown, L. Easy, S. Farley, B. Friedman, E. Grinaker, O. Izacard, I. Joseph, M. Kim, M. Leconte, J. Leddy, M. Løiten, C. Ma, J. Madsen, D. Meyerson, P. Naylor, S. Myers, J. Omotani, T. Rhee, J. Sauppe, K. Savage, H. Seto, B. Shanahan, M. Thomas, S. Tiwari, M. Umansky, N. Walkden, L. Wang, Z. Wang, P. Xi, T. Xia, X. Xu, H. Zhang, A. Bokshi, H. Muhammed, M. Estarellas, F. Riva, A. Fisher, C. MacMackin, E. Medwedeff, J. Holger, K.S. Kang, X. Xu, Y. Qin, Bout++, https://doi.org/10.5281/zenodo.7603558, Feb. 2023.

[55] B. Dudson, P. Hill, D. Dickinson, J. Parker, A. Allen, G. Breyiannis, J. Brown, L. Easy, S. Farley, B. Friedman, E. Grinaker, O. Izacard, I. Joseph, M. Kim, M. Leconte, J. Leddy, M. Løiten, C. Ma, J. Madsen, D. Meyerson, P. Naylor, S. Myers, J. Omotani, T. Rhee, J. Sauppe, K. Savage, H. Seto, D. Schwörer, B. Shanahan, M. Thomas, S. Tiwari, M. Umansky, N. Walkden, L. Wang, Z. Wang, P. Xi, T. Xia, X. Xu, H. Zhang, Bout++ v4.0.0, https://github.com/boutproject/BOUT-dev/tree/v4.0.0, Feb. 2017.

[56] M.A. Beer, S.C. Cowley, G.W. Hammett, Field-aligned coordinates for nonlinear simulations of tokamak turbulence, Phys. Plasmas 2 (7) (1995) 2687–2700, https://doi.org/10.1063/1.871232, arXiv:https://pubs.aip.org/aip/pop/article-pdf/2/7/2687/12714947/2687_1_online.pdf.

[57] X.Q. Xu, R.H. Cohen, T.D. Rognlien, J.R. Myra, Low-to-high confinement transition simulations in divertor geometry, Phys. Plasmas 7 (5) (2000) 1951–1958, https://doi.org/10.1063/1.874044, arXiv:https://pubs.aip.org/aip/pop/article-pdf/7/5/1951/12332239/1951_1_online.pdf.

[58] J.T. Omotani, B. Dudson, P. Hill, hypnotoad, https://doi.org/10.5281/zenodo.6360327, Apr. 2020.

[59] B. Dudson, P. Hill, D. Dickinson, J. Parker, A. Allen, G. Breyiannis, J. Brown, L. Easy, S. Farley, B. Friedman, E. Grinaker, O. Izacard, I. Joseph, M. Kim, M. Leconte, J. Leddy, M. Løiten, C. Ma, J. Madsen, D. Meyerson, P. Naylor, S. Myers, J. Omotani, T. Rhee, J. Sauppe, K. Savage, H. Seto, D. Schwörer, B. Shanahan, M. Thomas, S. Tiwari, M. Umansky, N. Walkden, L. Wang, Z. Wang, P. Xi, T. Xia, X. Xu, H. Zhang, Bout++ v4.2.0, https://doi.org/10.5281/zenodo.1464133, Oct. 2018.

[60] B. Dudson, P. Hill, D. Dickinson, J. Parker, A. Allen, G. Breyiannis, J. Brown, L. Easy, S. Farley, B. Friedman, E. Grinaker, O. Izacard, I. Joseph, M. Kim, M. Leconte, J. Leddy, M. Løiten, C. Ma, J. Madsen, D. Meyerson, P. Naylor, S. Myers, J. Omotani, T. Rhee, J. Sauppe, K. Savage, H. Seto, D. Schwörer, B. Shanahan, M. Thomas, S. Tiwari, M. Umansky, N. Walkden, L. Wang, Z. Wang, P. Xi, T. Xia, X. Xu, H. Zhang, A. Bokshi, H. Muhammed, M. Estarellas, Bout++ v4.3.0, https://doi.org/10.5281/zenodo.3518905, Oct. 2019.

[61] A.M. Dimits, Fluid simulations of tokamak turbulence in quasiballooning coordinates, Phys. Rev. E 48 (1993) 4070–4079, https://doi.org/10.1103/PhysRevE.48.4070, https://link.aps.org/doi/10.1103/PhysRevE.48.4070.

[62] B. Scott, Shifted metric procedure for flux tube treatments of toroidal geometry: avoiding grid deformation, Phys. Plasmas 8 (2) (2001) 447–458, https://doi.org/10.1063/1.1335832.

[63] B.D. Dudson, S.L. Newton, J.T. Omotani, J. Birch, On Ohm's law in reduced plasma fluid models, Plasma Phys. Control. Fusion 63 (12) (2021) 125008, https://doi.org/10.1088/1361-6587/ac2af9.

[64] J.R. Angus, S.I. Krasheninnikov, Inviscid evolution of large amplitude filaments in a uniform gravity field, Phys. Plasmas 21 (11) (2014) 112504, https://doi.org/10.1063/1.4901237.

[65] P. Stangeby, The Plasma Boundary of Magnetic Fusion Devices, Taylor & Francis, 2000.

[66] W. Fundamenski, O. Garcia, V. Naulin, R. Pitts, A. Nielsen, J.J. Rasmussen, J. Horacek, J. Graves, JET EFDA contributors, dissipative processes in interchange driven scrape-off layer turbulence, Nucl. Fusion 47 (5) (2007) 417–433, https://doi.org/10.1088/0029-5515/47/5/006.

[67] S. Patankar, Numerical Heat Transfer and Fluid Flow, Hemisphere Publishing Corporation, 1980.

[68] A. Arakawa, Computational design for long-term numerical integration of the equations of fluid motion: two-dimensional incompressible flow. Part i, J. Comput. Phys. 1 (1) (1966) 119–143, https://doi.org/10.1016/0021-9991(66)90015-5, https://www.sciencedirect.com/science/article/pii/0021999166900155.

[69] T.M. Austin, M. Berndt, J.D. Moulton, A memory efficient parallel tridiagonal solver, Preprint LA-VR-03-4149, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.9454&rep=rep1&type=pdf, 2004.

[70] M.L. Magnussen, Global numerical modeling of magnetized plasma in a linear device, Ph.D. thesis, Department of Physics, Technical University of Denmark, 2017, https://orbit.dtu.dk/en/publications/global-numerical-modeling-of-magnetized-plasma-in-a-linear-device.

[71] S. Balay, W.D. Gropp, L.C. McInnes, B.F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: E. Arge, A.M. Bruaset, H.P. Langtangen (Eds.), Modern Software Tools in Scientific Computing, Birkhäuser Press, 1997, pp. 163–202.

[72] S. Balay, S. Abhyankar, M.F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, W.D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M.G. Knepley, F. Kong, S. Kruger, D.A. May, L.C. McInnes, R.T. Mills, L. Mitchell, T. Munson, J.E. Roman, K. Rupp, P. Sanan, J. Sarich, B.F. Smith, S. Zampini, H. Zhang, H. Zhang, J. Zhang, PETSc/TAO users manual, Tech. Rep. ANL-21/39 - Revision 3.17, Argonne National Laboratory, 2022, https://petsc.org/release/docs/manual/manual.pdf.

[73] R.D. Falgout, hypre, Computer Software, https://doi.org/10.11578/dc.20190715.1, May 2019, https://github.com/hypre-space/hypre/releases/tag/v2.23.0.

[74] A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, C.S. Woodward, SUNDIALS: suite of nonlinear and differential/algebraic equation solvers, ACM Trans. Math. Softw. 31 (3) (2005) 363–396, https://doi.org/10.1145/1089014.1089020.

[75] B. Dudson, S. Farley, L.C. McInnes, Improved nonlinear solvers in bout++, arXiv preprint arXiv:1209.2054, 2012.

[76] W. Gracias, P. Tamain, E. Serre, R. Pitts, L. García, The impact of magnetic shear on the dynamics of a seeded 3d filament in slab geometry, in: Proceedings of the 22nd International Conference on Plasma Surface Interactions 2016, 22nd PSI, Nucl. Mater. Energy 12 (2017) 798–807, https://doi.org/10.1016/j.nme.2017.02.022, https://www.sciencedirect.com/science/article/pii/S2352179116302800.

[77] J.T. Omotani, BoutFastOutput, https://github.com/johnomotani/BoutFastOutput, Dec. 2019.

[78] G. Beckett, J. Beech-Brandt, K. Leach, Z. Payne, A. Simpson, L. Smith, A. Turner, A. Whiting, ARCHER2 service description, https://doi.org/10.5281/zenodo.14507040, 2024.

[79] T. Nicholas, J.T. Omotani, D. Bold, G. Decristoforo, B.D. Dudson, R. Doyle, P. Hill, xBOUT, https://doi.org/10.5281/zenodo.6945613, Aug. 2022, https://github.com/boutproject/xBOUT.

[80] S. Hoyer, J. Hamman, xarray: N-D labeled arrays and datasets in Python, J. Open Res. Softw. 5 (1) (2017), https://doi.org/10.5334/jors.148.

[81] S. Hoyer, M. Roos, H. Joseph, J. Magin, D. Cherian, C. Fitzgerald, M. Hauser, K. Fujii, F. Maussion, G. Imperiale, S. Clark, A. Kleeman, T. Nicholas, T. Kluyver, J. Westling, J. Munroe, A. Amici, A. Barghini, A. Banihirwe, R. Bell, Z. Hatfield-Dodds, R. Abernathey, B. Bovy, J. Omotani, K. Mühlbauer, M.K. Roszko, P.J. Wolfram, xarray, https://doi.org/10.5281/zenodo.598201, Jul. 2022, https://github.com/pydata/xarray.

[82] ISO/IEC 9834-8:2014, Information technology — Procedures for the operation of object identifier registration authorities — Part 8: Generation of universally unique identifiers (UUIDs) and their use in object identifiers, Standard, International Organization for Standardization, Geneva, CH, Aug. 2014, https://www.iso.org/standard/62795.html.