A Multigrid Algorithm for Nonlocal Collisional Electrostatic Drift-Wave Turbulence

John C. Bowman
Department of Mathematical Sciences, University of Alberta, Edmonton, Alberta, Canada T6G 2G1

A. Zeiler and D. Biskamp
Max-Planck-Institut für Plasmaphysik, EURATOM Association, D 85748 Garching, Germany
(September 18, 1999)

We have developed a three-dimensional anisotropic multigrid solver for simulating nonlocal collisional electrostatic drift-wave turbulence in a tokamak with magnetic shear. As an example, the solver has been used to obtain entire flux-surface solutions of the nonlocal Hasegawa–Wakatani equations in the absence of curvature effects. The implicit treatment of the parallel-gradient terms permits the use of a relatively large time step. Considerable effort was made in the design of the implicit solver to ensure that the presence of anisotropy does not lead to a significant degradation in performance. The multigrid algorithm has several advantages over a pseudospectral Poisson solver; most importantly, all nonlinear terms, including those in the Ohm's law, can be retained in a straightforward manner. Although in this work the solver is illustrated using straightened tokamak (sheared slab) geometry, the object-oriented construction of the code will facilitate the eventual inclusion of curvature terms and the complete nonlinear reduced Braginskii equations, including ion thermal dynamics.

I. INTRODUCTION

Tokamaks are experimental toroidal devices for studying the feasibility of controlled thermonuclear fusion as a relatively clean and abundant energy source for the future. They use a combination of magnetic and electric fields to confine plasmas at very high temperatures until thermonuclear fusion of the nuclei occurs. One of the major obstacles in demonstrating the scientific feasibility of this technology is the dramatic reduction in confinement that results from the heat and particle transport associated with small-scale turbulent fluctuations. The goal of modern plasma turbulence theory is to understand and ultimately control this *anomalous transport*.

In this work we describe a three-dimensional simulation of collisional electrostatic drift waves [1,2]. These instabilities are thought to play a important role in the outer edge regions of tokamaks, where the temperature is low enough for collisionality to dominate. The equations that describe drift-waves are nonlocal (the coefficients of the partial derivatives vary in space) so that discrete Fourier transform methods cannot be used to invert the Laplace operator that arises. The curvilinear geometry, anisotropy, and the sheared magnetic field introduce further complications, all of which may be handled in a natural way with a multigrid solver. The problem of computing turbulent transport in a tokamak reactor thus provides an excellent example of the advantages of multigrid methods over conventional spectral methods for evolving systems of parabolic differential equations in time.

We begin in Sect. II with a general discussion of the geometry, equations, and boundary conditions for the Hasegawa–Wakatani drift-wave model. Then, in Sect III, we describe and benchmark our multigrid algorithm against a pseudospectral Poisson solver for a local version of the equations. Nonlocal simulation results of our multigrid code, in the absence of magnetic shear, are presented in Sect. IV. Finally, in Sect. V, we present a simulation of a full flux surface for a straightened tokamak with magnetic shear. We conclude with some final remarks in Sect. VI.

II. HASEGAWA–WAKATANI MODEL

A. Coordinate system

The volume of the tokamak edge region to be simulated lies between two torii of different minor radii (the volume of revolution of a poloidal annulus about the major axis). It is convenient to introduce a transformation that maps this toroidal geometry into a rectilinear region described by Cartesian coordinates. In this straightened system, new terms representing curvature effects will then arise in the equations of motion. For simplicity these geometrical effects are neglected in this work; however, the eventual inclusion of curvature effects and magnetic field gradients, both crucial to physics of the ballooning-mode [3,4], should be relatively straightforward.

In the straightened geometry, we represent the minor radius coordinate by x, the poloidal direction by y, and the toroidal direction by z. Tokamak magnetic fields are characterized by both poloidal and toroidal components, so that the magnetic field lines twist around the surface of the torus. The amount of twist depends on the radial coordinate x; the resulting *magnetic shear* is an important damping mechanism for drift-wave turbulence. Since the dynamics of drift-wave turbulence tends to vary only weakly in the direction of the magnetic field, it is numerically advantageous to introduce a new coordinate system (x', y', z') that is aligned with the magnetic field direction instead of the toroidal direction, such that z' is always parallel to the local magnetic field $\mathbf{B}$ [4–7]. For the case of the prototypical sheared field

$$B = B_0(\hat{z} - \alpha x \hat{y}), \tag{1}$$

the appropriate transformation is

$$x' = x, \tag{2a}$$

$$y' = y + \alpha z x, \tag{2b}$$

$$z' = z. \tag{2c}$$

Upon denoting (x', y', z') by (x^0, x^1, x^2), the contravariant basis vectors $\mathbf{e}_i = \partial \mathbf{r}/\partial x^i$ corresponding to the coordinate transformation Eq. (2) may be expressed as

$$\mathbf{e}_0 = \hat{x} - \alpha z \hat{y}, \tag{3a}$$

$$\mathbf{e}_1 = \hat{y}, \tag{3b}$$

$$\mathbf{e}_2 = \hat{z} - \alpha x \hat{y}, \tag{3c}$$

where $\hat{x}$, $\hat{y}$, and $\hat{z}$ are the Cartesian unit vectors in the x, y, and z directions, respectively. The numerical discretizations are then performed with respect to the computational coordinates (x', y', z'), using the contravariant basis $(\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2)$.

Note that $\mathbf{e}_2$ is in the direction of the magnetic field, so that $\mathbf{B} = B\mathbf{e}_2$. The other two basis vectors lie in a poloidal plane. Since the thickness of the (straightened) annulus in x is much smaller than its extension in z, the dominant effect of magnetic shear in Eq. (3) arises from the αz term. As is common practice, we therefore adopt the simplified transformation

$$\mathbf{e}_0 = \hat{x} - \alpha z \hat{y}, \tag{4a}$$

$$\mathbf{e}_1 = \hat{y}, \tag{4b}$$

$$\mathbf{e}_2 = \hat{z}, \tag{4c}$$

thereby capturing the lowest-order effect of the shear (in the inverse-aspect ratio). The corresponding metric tensor, with the covariant components $g_{ij} = \mathbf{e}_i \cdot \mathbf{e}_j$, then reduces to

$$\boldsymbol{g} = \begin{pmatrix} 1 + \alpha^2 z^2 & \alpha z & 0 \\ \alpha z & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{5}$$

The inverse metric tensor, with components g^{ij}, is

$$\boldsymbol{g}^{-1} = \begin{pmatrix} 1 & -\alpha z & 0 \\ -\alpha z & 1 + \alpha^2 z^2 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{6}$$

The covariant basis vectors $\mathbf{e}^i = g^{ij}\mathbf{e}_j$ (summing over repeated indices) are found to be

$$\mathbf{e}^0 = \hat{x}, \tag{7a}$$

$$\mathbf{e}^1 = \hat{y} - \alpha z \hat{x}, \tag{7b}$$

$$\mathbf{e}^2 = \hat{z}. \tag{7c}$$

We may now compute the covariant derivative $\nabla \doteq \mathbf{e}^i \partial/\partial x'^i$, (the notation $\doteq$ is used to emphasize definitions)

$$\nabla \doteq \hat{x}\left(\frac{\partial}{\partial x'} + \alpha z \frac{\partial}{\partial y'}\right) + \hat{y}\frac{\partial}{\partial y'} + \hat{z}\frac{\partial}{\partial z'}, \tag{8}$$

of which the part perpendicular to $\hat{z}$ is just

$$\nabla_\perp \doteq \hat{x}\left(\frac{\partial}{\partial x'} + \alpha z \frac{\partial}{\partial y'}\right) + \hat{y}\frac{\partial}{\partial y'}, \tag{9}$$

or, equivalently,

$$\nabla_\perp \doteq \mathbf{e}_0\left(\frac{\partial}{\partial x'} + \alpha z \frac{\partial}{\partial y'}\right) + \mathbf{e}_1\left[\alpha z \frac{\partial}{\partial x'} + (1+\alpha^2 z^2)\frac{\partial}{\partial y'}\right]. \tag{10}$$

We also define the component of the gradient parallel to $\hat{z}$, $\nabla_\parallel \doteq \partial/\partial z'$.

The divergence of a vector $\mathbf{A}$ may be expressed generally as

$$\nabla \cdot \mathbf{A} \doteq g^{-1/2}\frac{\partial}{\partial x^i}(g^{1/2}A^i), \tag{11}$$

where in our case $g = \det|\boldsymbol{g}|$ evaluates to 1. The perpendicular and parallel divergences $\nabla_\perp \cdot \mathbf{A}$ and $\nabla_\parallel \cdot \mathbf{A}$, respectively, are thus

$$\nabla_\perp \cdot \mathbf{A} \doteq \frac{\partial}{\partial x'}A^0 + \frac{\partial}{\partial y'}A^1, \tag{12a}$$

$$\nabla_\parallel \cdot \mathbf{A} \doteq \frac{\partial}{\partial z'}A^2. \tag{12b}$$

B. Nonlocal equations

For nonlocal simulation of resistive drift-wave turbulence, we normalize the coordinates (x, y, z, t) to $(\rho_s, \rho_s, L_\parallel, \Omega_i^{-1})$ and the total potential and density fields (ϕ, n) to $(T_e/e, \bar{n})$. Here $\rho_s = c_s/\Omega_i$, $\Omega_i = eB/(m_i c)$, $c_s = (T_e/m_i)^{1/2}$, T_e is the electron temperature, m_i is the ion mass, $L_\parallel = \rho_s[B/(ec\eta_\parallel \bar{n})]^{1/2}$, and $\bar{n}$ is some characteristic density. In this normalization, the coupled set of equations for the potential and density studied by Hasegawa and Wakatani [1] appear as

$$\nabla_\perp \cdot \left(n\frac{d}{dt}\nabla_\perp \phi\right) + \nabla_\parallel \cdot \left(\nabla_\parallel \phi - \frac{\nabla_\parallel n}{n}\right) = D_\phi \nabla_\perp{}^4 \phi, \tag{13a}$$

$$\frac{dn}{dt} + \nabla_\parallel \cdot \left(\nabla_\parallel \phi - \frac{\nabla_\parallel n}{n}\right) = D_n \nabla_\perp{}^2 n. \tag{13b}$$

where

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \hat{z}\times\nabla\phi \cdot \nabla. \tag{14}$$

To evolve Eqs. (13) in the twisted coordinate system (2) we need to compute the density-weighted Laplacian

$$\nabla_\perp \cdot (n\nabla_\perp \phi) \doteq \left(\frac{\partial}{\partial x'} + \alpha z \frac{\partial}{\partial y'}\right) n \left(\frac{\partial}{\partial x'} + \alpha z \frac{\partial}{\partial y'}\right)\phi + \frac{\partial}{\partial y'} n \frac{\partial}{\partial y'}\phi \tag{15}$$

and the $\mathbf{E}\times\mathbf{B}$ advection velocity

$$\mathbf{v} \doteq \hat{z}\times\nabla\phi = \mathbf{e}_1\left(\frac{\partial\phi}{\partial x'} + \alpha z \frac{\partial\phi}{\partial y'}\right) - (\mathbf{e}_0 + \alpha z \mathbf{e}_1)\frac{\partial\phi}{\partial y'} = -\mathbf{e}_0\frac{\partial\phi}{\partial y'} + \mathbf{e}_1\frac{\partial\phi}{\partial x'}. \tag{16}$$

The form of the latter result is anticipated, in view of the scalar invariance of the advective nonlinearity $\mathbf{v}\cdot\nabla$ under linear transformation.

C. Boundary conditions

In straightened tokamak coordinates, we impose a Dirichlet condition on the density $n = \hat{n}(x, y, z)$,

$$\hat{n}(x_{\min}, y, z) = n_{\max}, \qquad \hat{n}(x_{\max}, y, z) = n_{\min} \qquad \forall\ y, z. \tag{17}$$

This prescription appears also as a Dirichlet condition if we express $n = n(x', y', z')$ in the twisted coordinates (2),

$$n(x'_{\min}, y', z') = n_{\max}, \qquad n(x'_{\max}, y', z') = n_{\min} \qquad \forall\ y', z'. \tag{18}$$

In imposing these density values at the x' boundaries, one must be careful to avoid introducing artificially large density gradients. This can be accomplished by adopting a boundary condition on ϕ that is compatible with the particle fluxes required to maintain the imposed density differential. Upon noting that Eq. (18) is consistent with the requirement of zero $\mathbf{E}\times\mathbf{B}$ advection in the direction $\mathbf{e}_1$ $(= \hat{y})$ at the x' boundaries, we stipulate that $\mathbf{v}\cdot\mathbf{e}^1 = 0$ at both boundaries, so that the advection at the boundaries is purely in the $\mathbf{e}_0$ direction. Equation (16) then requires that

$$\frac{\partial}{\partial x'}\phi(x'_{\min}, y', z') = 0, \qquad \frac{\partial}{\partial x'}\phi(x'_{\max}, y', z') = 0 \qquad \forall\ y', z'. \tag{19}$$

A self-consistent $\mathbf{E}\times\mathbf{B}$ velocity (physically responsible for maintaining the imposed density profile) then arises naturally from this Neumann boundary condition on ϕ in the x' direction.[1]

In y and z, the boundary conditions on the state vector $\hat{\mathbf{u}} \doteq (\phi, n)$ are doubly periodic:

$$\hat{\mathbf{u}}(x, y_{\min}, z) = \hat{\mathbf{u}}(x, y_{\max}, z) \qquad \forall\ x, z \tag{20}$$

and

$$\hat{\mathbf{u}}(x, y, z_{\min}) = \hat{\mathbf{u}}(x, y, z_{\max}) \qquad \forall\ x, y. \tag{21}$$

Upon expressing $\hat{\mathbf{u}}(x, y, z) = \mathbf{u}(x', y', z')$ the transformed boundary conditions are found to be [4–6, 8]

$$\mathbf{u}(x', y'_{\min}, z') = \mathbf{u}(x', y'_{\max}, z') \qquad \forall\ x', z' \tag{22}$$

and

$$\mathbf{u}(x', y', z'_{\min}) = \mathbf{u}(x', y' + \alpha x'(z'_{\max} - z'_{\min}), z'_{\max}) \qquad \forall\ x', y'. \tag{23}$$

In the numerical implementation, linear interpolation is used to implement the parallel boundary condition (23).

D. Local approximation

In many simulations of the Hasegawa–Wakatani equations, the restriction $n = 1 + L_n^{-1}(x - x_0) + \widetilde{n}$ (where x_0 is the x-coordinate of the center of the box, $x - x_0 \ll L_n$, and $\widetilde{n} \ll 1$) is imposed to allow Eqs. (13) to be approximated by

$$\frac{d}{dt}\nabla_\perp{}^2\phi + \nabla_\parallel{}^2(\phi - \widetilde{n}) = D_\phi \nabla_\perp{}^4\phi, \tag{24a}$$

$$\frac{d\widetilde{n}}{dt} + \nabla_\parallel{}^2(\phi - \widetilde{n}) = D_n \nabla_\perp{}^2\widetilde{n} + L_n^{-1}\phi_y. \tag{24b}$$

The code described in the following section permits the solution of the full equations (13), allowing it to be used to obtain self-consistent turbulence-driven profiles. However, the extra terms in Eqs. (13) not present in Eqs. (24) may be switched off (and the diamagnetic term $L_n^{-1}\phi_y$ enabled) to facilitate comparison with other local simulations.

---

[1] In practice, it is convenient in the implicit solver discussed below to use a Dirichlet condition in the x' direction for both ϕ and n. The Neumann boundary on ϕ is enforced only when evaluating the perpendicular derivatives in the nonlinear source routine; the computed boundary values are then used to satisfy the Dirichlet condition required by the implicit solver during subsequent time steps.

III. NUMERICAL IMPLEMENTATION

The coupled equations Eq. (13) are solved as an initial value problem, using a general-purpose object-oriented code called `Triad`, written in C^{++}. The kernel routines of this code implement operations that are common to a wide class of dynamical problems, for example, parameter input, parsing, generic integration algorithms, dynamic-time step adjustment, a restart facility, and error handling. Further details of `Triad` will be described elsewhere.

A. Finite differencing

For the purposes of numerical discretization, Eqs. (13) may be rewritten in a flux-conservative form by exploiting the solenoidal nature of the $\mathbf{E}\times\mathbf{B}$ velocity $\mathbf{v} \doteq \hat{z}\times\nabla\phi$,

$$\nabla_\perp \cdot \left(n \frac{\partial}{\partial t} \nabla_\perp \phi \right) + \nabla_\perp \cdot [n \nabla_\perp \cdot (\mathbf{v} \nabla_\perp \phi)] + \nabla_\parallel \cdot \left(\nabla_\parallel \phi - \frac{\nabla_\parallel n}{n} \right) = D_\phi {\nabla_\perp}^4 \phi - H_\mathbf{v}\{{\nabla_\perp}^2 \phi\}, \tag{25a}$$

$$\frac{\partial n}{\partial t} + \nabla_\perp \cdot (\mathbf{v} n) + \nabla_\parallel \cdot \left(\nabla_\parallel \phi - \frac{\nabla_\parallel n}{n} \right) = D_n {\nabla_\perp}^2 n - H_\mathbf{v}\{n\}. \tag{25b}$$

Besides having better conservation properties, this form can be expressed relatively simply in our twisted coordinate system, with the effect of shear entering explicitly only through the $\nabla_\perp$ operator, given in contravariant form by Eq. (10).

The velocity-dependent hyperviscosity operator $H_\mathbf{v}$ introduced in Eqs. (25) minimizes the range of scales devoted to modeling small-scale dissipation. It is designed to absorb the directly cascading enstrophy and internal energy at the grid scale in a manner that emulates (in one-dimension) the stabilizing effect of an upwind scheme on convective equations. Following Ref. [3], we evaluate the spatial derivatives in Eq. (25) to fourth order and adopt the hyperviscosity operator

$$H_\mathbf{v} \doteq H(v_x, \Delta x, \frac{\partial}{\partial x}) + H(v_y, \Delta y, \frac{\partial}{\partial y}), \tag{26}$$

where

$$H(v_x, \Delta x, \frac{\partial}{\partial x}) = \mu \Delta x^3 \frac{\partial}{\partial x} |v_x| \frac{\partial^3}{\partial x^3}. \tag{27}$$

However, the hyperviscosity was implemented here to higher order than in Ref. [3], by writing it as a correction to the perpendicular fluxes. The third-order derivatives appearing in these flux corrections were evaluated to fourth-order using the formula

$$F'''(x) = \frac{1}{2\Delta^3}[-F(x-2\Delta) + 2F(x-\Delta) - 2F(x+\Delta) + F(x+2\Delta)]. \tag{28}$$

The value $\mu = 0.25$ was found to be sufficient to avoid nonlinear convective instability in all cases.

B. Time-stepping algorithms

The code `Triad` currently defines the following general-purpose integration algorithms: first-order Euler, a second-order Predictor–Corrector method, second-order Leapfrog, second-order Runge–Kutta, fourth-order Runge–Kutta, fifth-order Cash–Karp Runge–Kutta integrator. With the exception of the Euler method, all of these schemes supply a mechanism for dynamic adjustment of the time step based on an internal error estimate.

To avoid unnecessary restriction of the time step by the parallel-gradient terms in Eq. (13), we treat these terms implicitly with a trapezoidal approximation. Since this part of the scheme is only second-order accurate in the time step, it is reasonable to adopt a second-order scheme as the integrator for the perpendicular dynamics as well. Another reason for not using a higher-order integration algorithm is that the convergence of the anisotropic Poisson solver described in Sec. III C tends to improve as the coefficient of the parallel derivative, which is proportional to the time step, is reduced. The larger time step that would be afforded by a higher-order integration algorithm is therefore not necessarily desirable.

For pedagogical reasons, we begin the discussion of our semi-implicit time-stepping algorithm by considering the i^{th} time step, with size τ, of the Euler method applied to Eq. (13),

$$\begin{aligned}\nabla_\perp \cdot (n_{i-1}\nabla_\perp \phi_i) + \frac{\tau}{2}\nabla_\parallel \cdot \left(\nabla_\parallel \phi_i - \frac{\nabla_\parallel n_i}{n_i}\right) &= \nabla_\perp \cdot (n_{i-1}\nabla_\perp \phi_{i-1}) - \frac{\tau}{2}\nabla_\parallel \cdot \left(\nabla_\parallel \phi_{i-1} - \frac{\nabla_\parallel n_{i-1}}{n_{i-1}}\right) \\ &\quad -\tau \nabla_\perp \cdot (n_{i-1}\hat{z}\times\nabla\phi_{i-1} \cdot \nabla\nabla_\perp \phi_{i-1}) \\ &\quad +\tau D_\phi \nabla_\perp{}^4 \phi_{i-1}, \end{aligned} \tag{29a}$$

$$\begin{aligned} n_i + \frac{\tau}{2}\nabla_\parallel \cdot \left(\nabla_\parallel \phi_i - \frac{\nabla_\parallel n_i}{n_i}\right) &= n_{i-1} - \frac{\tau}{2}\nabla_\parallel \cdot \left(\nabla_\parallel \phi_{i-1} - \frac{\nabla_\parallel n_{i-1}}{n_{i-1}}\right) \\ &\quad -\tau \hat{z}\times\nabla\phi_{i-1} \cdot \nabla n_{i-1} + \tau D_\eta \nabla_\perp{}^2 \eta_{i-1}. \end{aligned} \tag{29b}$$

Upon defining $\mathbf{u} \doteq (\phi, n)$ and the transformation

$$\mathcal{T}(\tau)\mathbf{u} \doteq \begin{pmatrix} \nabla_\perp \cdot (n_{i-1}\nabla_\perp \phi) + \frac{\tau}{2}\nabla_\parallel \cdot \left(\nabla_\parallel \phi - n^{-1}\nabla_\parallel n\right) \\ n + \frac{\tau}{2}\nabla_\parallel \cdot \left(\nabla_\parallel \phi - n^{-1}\nabla_\parallel n\right) \end{pmatrix}, \tag{30}$$

we may write Eq. (29) in the compact form

$$\mathcal{T}(\tau)\mathbf{u}_i = \mathcal{T}(-\tau)\mathbf{u}_{i-1} + \tau \mathbf{S}_{i-1}, \tag{31}$$

where the advective nonlinearities, treated explicitly, are incorporated into the source $\mathbf{S}$, along with the perpendicular dissipative terms. A generic integration routine may thus be used to integrate Eq. (13) upon first transforming $\mathbf{u}_{i-1}$ to the new variable $\mathcal{T}(-\tau)\mathbf{u}_{i-1}$, applying the integration method, and finally transforming back with the inverse transformation $\mathcal{T}^{-1}(\tau)$. The implementation of this inversion operator is described in the next section.

The generalization of Eq. (31) to a second-order predictor–corrector scheme is straightforward:

$$\mathcal{T}(\tau)\widetilde{\mathbf{u}}_i = \mathcal{T}(-\tau)\mathbf{u}_{i-1} + \tau \mathbf{S}_{i-1}, \tag{32a}$$

$$\mathcal{T}(\tau)\mathbf{u}_i = \mathcal{T}(-\tau)\mathbf{u}_{i-1} + \frac{\tau}{2}(\mathbf{S}_{i-1} + \widetilde{\mathbf{S}}_i), \tag{32b}$$

where the source term $\widetilde{\mathbf{S}}_i$ is evaluated at the intermediate point $\widetilde{\mathbf{u}}_i$.[2] Normally, evaluation of the transformation $\mathcal{T}$ and $\mathcal{T}^{-1}$ are the most computationally expensive parts of the calculation. Other integration algorithms can be written in a similar manner, such as the following second-order leapfrog scheme:

$$\mathcal{T}(\tau)\widetilde{\mathbf{u}}_i = \mathcal{T}(-\tau)\widetilde{\mathbf{u}}_{i-1} + \tau \mathbf{S}_{i-1}, \tag{33a}$$

$$\mathcal{T}(\tau)\mathbf{u}_i = \mathcal{T}(-\tau)\mathbf{u}_{i-1} + \tau\widetilde{\mathbf{S}}_{i-1}. \tag{33b}$$

Note that the predictor–corrector algorithm will typically execute somewhat faster than the leapfrog scheme since it requires one less forward transformation $\mathcal{T}(-\tau)$ but the same number of inverse transformations $\mathcal{T}^{-1}(\tau)$.

All of the generic integration routines in `Triad` can in this manner be performed in a transformed space; the user need only supply the transformation routine and its inverse.

C. Anisotropic Poisson solver

The inverse transformation in the previously described time-stepping algorithm at the i^{th} time step can be accomplished by inverting the anisotropic elliptic operator

[2] For the corrector in Eq. (32b) to be strictly second-order, the n_{i-1} factor in Eq. (30) should be replaced by $(n_{i-1} + n_i)/2$; however, in practice the error introduced in Eq. (32b) is negligible.

$$\nabla_\perp \cdot (n_{i-1}\nabla_\perp \phi_i) + \frac{\tau}{2}\nabla_\parallel \cdot \left(\nabla_\parallel \phi_i - \frac{\nabla_\parallel n_i}{n_{i-1}}\right) = f_i^\phi, \tag{34a}$$

$$n_i + \frac{\tau}{2}\nabla_\parallel \cdot \left(\nabla_\parallel \phi_i - \frac{\nabla_\parallel n_i}{n_{i-1}}\right) = f_i^n; \tag{34b}$$

This operator, linear in n_i and ϕ_i, is obtained by added the small quantity $\tau\nabla_\parallel \cdot [(n_i^{-1} - n_{i-1}^{-1})\nabla_\parallel n_i]/2$ to both sides of each equation in (29). (In practice, this small correction to the explicitly treated source term can simply be ignored.)

Equations (34) are inverted with an anisotropic multigrid solver. This solver, which is based on an xy-zebra-surface Gauss–Seidel smoother (described in Appendix A), in turn requires the solution of a tridiagonal equation for n_i and an equation for ϕ_i of the form

$$[\nabla_\perp \cdot (n_{i-1}\nabla_\perp) + \epsilon]\,\phi_i = f_i, \tag{35}$$

where ϵ is a factor proportional to the time step (the term $\epsilon\phi_i$ corresponds in Eq. (A4) to the central term $-2\phi_{i,j}$ that is moved to the left-hand side and treated implicitly). Solutions of this 2D anisotropic Poisson-like equation are obtained with a secondary multigrid solver based on an y-zebra-line Gauss–Seidel tridiagonal smoother, discussed in more detail in Appendix A and in Refs. [9–11].

A single V-cycle iteration of the full xy-zebra-surface Gauss–Seidel multigrid solver typically reduces the root-mean-squared *defect* (residual) by a factor of 4 or 5. Because the solver is initialized with the values computed during the previous time step, we find in practice that two iterations of the solver are sufficient to yield an accurate dynamical evolution of the Hasegawa–Wakatani equations. This was tested by comparison with the solutions obtained (i) using many multigrid iterations per time step and (ii) for the local equations, using the pseudospectral solver described in the following subsection.

The results of the second test are depicted in Figs. 1 and 2 for a $64 \times 32 \times 16$ grid corresponding to a physical domain with dimensions $36\rho_s \times 18\rho_s \times 38L_\parallel$, using the dissipation parameters $D_\phi = D_n = 10^{-3}$. We compare the total turbulent energy density $E = \left\langle (\nabla_\perp \phi)^2 + \widetilde{n}^2 \right\rangle$ and particle flux $\Gamma = -\langle n\partial\phi/\partial y\rangle$, where $\langle\cdot\rangle$ denotes a volume average, for identical pseudospectral and multigrid runs, in the absence of magnetic shear and using periodic boundary conditions in all three directions. We also depict both local and nonlocal multigrid versions of these same runs using the Neumann/Dirichlet conditions (18) and (19) in the x direction. The chosen density scale length, $L_n = 34\rho_s$, satisfies the local approximation only marginally and for this reason the nonlocal simulations depicted in Figs. 1 and 2 depart from the other three runs, which assume locality. One sees that even for small initial conditions (random values chosen uniformly from the interval $[-10^{-3}, 10^{-3}]$), the variation with x of the coefficient n^{-1} in the Ohm's Law term of Eq. (13b) has a significant effect on the linear dynamical evolution.

D. Pseudospectral solver

Testing and benchmarking of the multigrid solver was facilitated with a pseudospectral option, which uses fast Fourier transforms (FFT's) to invert the linear operator in Eqs. (24). This option is available only for the local equations. The nonlinear and diamagnetic terms, however, are computed with finite differencing, just as with the multigrid solver; this avoids the need to introduce de-aliasing points in the FFT buffers and keeps the FFT sizes as small as possible. The linear dissipation terms may be computed either in the spatial domain, by finite-differencing—just as with the multigrid solver—or by multiplication in the Fourier-transformed domain.

E. Near-singular operators

With periodic boundary conditions in x and y, the operator (35) is singular when $\epsilon = 0$; the solution for ϕ is then determined only up to an arbitrary function of z. It is therefore not surprising that for these boundary conditions the efficiency of the multigrid algorithm dramatically degrades as $\epsilon \to 0$. However, there is a straightforward procedure to add the correct z-dependent solution to the solver result. We illustrate the procedure under the local approximation (for our application this is the only case where periodic boundary conditions in x are relevant).

Let us rewrite the local version of Eq. (34) (replacing n_{i-1} by unity) as

$$\mathcal{L}\begin{pmatrix}\phi\\ n\end{pmatrix} = \begin{pmatrix}f_\phi\\ f_n\end{pmatrix}. \tag{36}$$

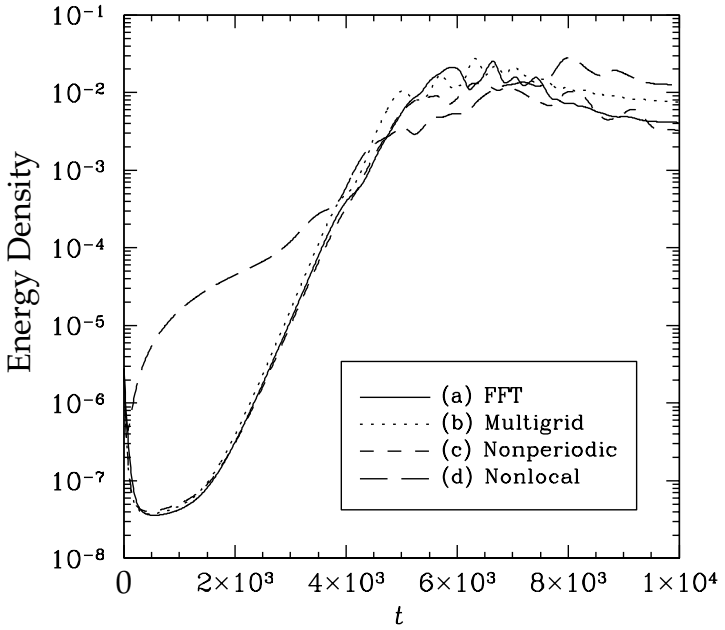

FIG. 1. Comparison of the total energy density E for identical (a) pseudospectral and (b) multigrid runs, with periodic boundary conditions, as well as for (c) local and (d) nonlocal multigrid runs using the boundary conditions (18) and (19) in the x direction.

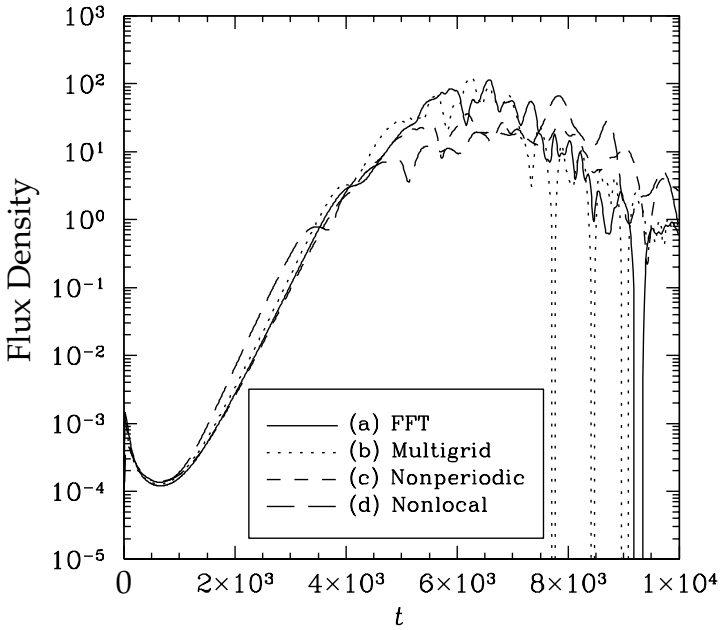

FIG. 2. Comparison of the flux density Γ for the runs in Fig. 1.

Because of the near-singularity of the perpendicular part of this operator, the solver returns a solution (ξ, η) approximating not the exact solution (ϕ, n) but the contaminated solution $(\phi + g(z), n + h(z))$. At each level of the multigrid hierarchy, the unknown functions $g(z)$ and $h(z)$ may be computed by applying $\mathcal{L}$ to the solver result and averaging over x and y:

$$\left\langle \mathcal{L} \begin{pmatrix} \xi \\ \eta \end{pmatrix} \right\rangle_{xy} = \left\langle \begin{pmatrix} f_\phi \\ f_n \end{pmatrix} \right\rangle_{xy} + \begin{pmatrix} \frac{1}{2}\tau {\nabla_\parallel}^2 (g - h) \\ h + \frac{1}{2}\tau {\nabla_\parallel}^2 (g - h) \end{pmatrix}, \tag{37}$$

which, in terms of the defect $(d_\phi, d_n) \doteq \mathcal{L}(\xi, \eta) - (f_\phi, f_n)$, leads to the equations

$$h = \langle d_n - d_\phi \rangle_{xy}\,, \tag{38a}$$

$$\frac{\tau}{2} {\nabla_\parallel}^2 (g - h) = \langle d_\phi \rangle_{xy}\,. \tag{38b}$$

Evaluation of g requires the inversion of a 1-dimensional Poisson equation, which is readily accomplished with a tridiagonal solver. Upon subtracting the correction (g, h) from the solver result, one obtains an iteration that converges to the true solution (ϕ, n).

F. Performance

Our algorithm has distinct advantages relative to a fully pseudospectral Poisson solver. While on a scalar machine the computation time for a single iteration[3] of the multigrid solver applied to Eqs. (24) is about the same as for a pseudospectral code, a multigrid solver should parallelize more effectively over a distributed memory architecture. This is because the implementation of a multidimensional fast Fourier transform on a distributed memory system effectively requires a matrix transposition, entailing high communication costs. A multigrid algorithm can also handle more realistic boundary conditions, such as Eqs. (18), (19), and (23). Furthermore, all nonlinear terms can be retained in a straightforward manner; in contrast, pseudospectral solvers require linearization of the n_{i-1} factor appearing in Eq. (35). Finally, we mention that for a large number of modes N, the scaling of the multigrid method, $\mathcal{O}(N)$, is formally better than that of the fast Fourier transform, $\mathcal{O}(N \ln N)$.

While the execution time for a single time step of our semi-implicit algorithm is not substantially greater than that for an explicit code (based on a 2D Poisson solver), we have found that the implicit treatment of the parallel-gradient terms permits a larger time step (typically more than a factor of 10 larger). For the nonlocal reference case depicted in Figs. 1 and 2, we compared the results obtained with explicit and implicit treatment of the parallel terms. We found no significant differences in the evolution of the flux and energy density. The fact that the explicit scheme required a time step 25 times smaller than that required by the implicit method is a signature of the existence of a strongly damped mode. Instead of exactly resolving the decay of this mode, the implicit algorithm makes sure that it never gets excited.

In the interest of easy program maintenance and reusable code, the object-oriented C^{++} programming language appeared to be an excellent choice for this project. The performance of this language was carefully evaluated before proceeding. It was found that, if certain programming practices were adhered to (such as avoiding constructor calls and dynamic memory allocation in the middle of loops), the performance of C^{++} can equal that of code written in Fortran. For example, an efficient array class written in C^{++} was used to compute a Laplacian; the speed of the routine was found to be within 8% of the performance of optimized Fortran-77 code. Time profiling of `Triad` showed that by far the most computationally expensive part of the code is the computation of a modified 3-dimensional Laplacian operator in the defect routine of the primary multigrid solver. By benchmarking this time-critical section of code in both C^{++} and Fortran, we were able to establish that there is no significant loss of performance in using C^{++} for this application. In a few places, it was nevertheless necessary to hand-optimize operations on user-defined objects by expressing them explicitly in terms of operations on their components (e.g., writing a complex equation explicitly in terms of its real and imaginary parts). The origin of this problem is that most existing C^{++} compilers currently write intermediate user-defined objects to memory instead of retaining them where possible in CPU registers.[4]

---

[3] One or two iterations of the solver is normally adequate, as described in Sect. III C.

[4] One exception to this is the KAI C^{++} to C *translator*, which has been designed for high-performance computing and is

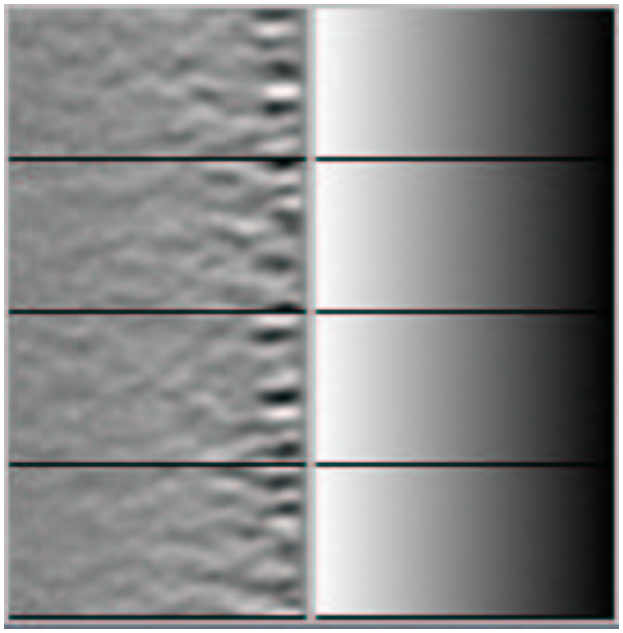

FIG. 3. Nonlocal simulation of ϕ (left) and n (right) at $t = 1000\Omega_i$. Black denotes low values.

IV. NONLOCAL SIMULATION

In Figs. 3–5 we illustrate typical xy-cross sections of the electrostatic potential and density that were obtained with the multigrid solver applied to the Hasegawa–Wakatani model in the absence of magnetic shear. These snapshots correspond to the linear, turbulent, and final saturated sheared-flow evolutionary stages, respectively. The stacked frames correspond to four different z values. A $127 \times 64 \times 4$ grid was used to model a box of size $36\rho_s \times 18\rho_s \times 38L_\parallel$, using the boundary conditions of Sect. II C, the dissipation parameters $D_\phi = D_n = 10^{-4}$, and the density scale length and initial conditions of Sect. III C. The density and potential profiles at $t = 1.2 \times 10^4\Omega_i$ are depicted in Fig. 6.

V. FLUX SURFACE SIMULATION

Because of the enormous number of poloidal modes that are required for a full flux surface treatment of drift-wave turbulence in a tokamak, most previous simulations have been confined to a narrow flux-tube region enclosing a magnetic field line as it winds around the torus. Let B_T and B_P be the strengths of the toroidal and poloidal magnetic fields and R and a be the major and minor radii of the tokamak, respectively. If the ratio $q = aB_T/(RB_P)$ (for which the isosurfaces are flux surfaces) is a rational number m/n, then after following the field line m times around the toroidal direction, the field line will have made n poloidal circuits. To avoid following the field line the full periodicity length $2\pi mR$, flux-tube simulations follow the field line only for one poloidal circuit, a distance of $2\pi qR$, somewhat misleadingly known as the *connection length*. After one connection length the field line does not return to the same toroidal location (unless $n = 1$) and therefore, since the turbulence correlation length in the parallel direction is typically larger than the connection length (see Fig 8), a number of researchers [4, 6, 12] have introduced *extended flux-tube models*, in which the flux tube is followed further, a least a parallel correlation length. For example, Zeiler *et al.* followed the field line three connection lengths before reconnected the flux tube to itself [4]. In effect, they follow the field line for about ten toroidal circuits, whereas in a full flux surface simulation, only one toroidal circuit must be made.

---

available for many workstations and also for parallel computers like the Cray T3E. The authors hope that future improvements to native C^{++} compilers will allow class data to be retained in CPU registers, allowing time-critical sections of code to be expressed compactly, in accordance with the object-oriented philosophy of the C^{++} language and without the need for intermediate translators.

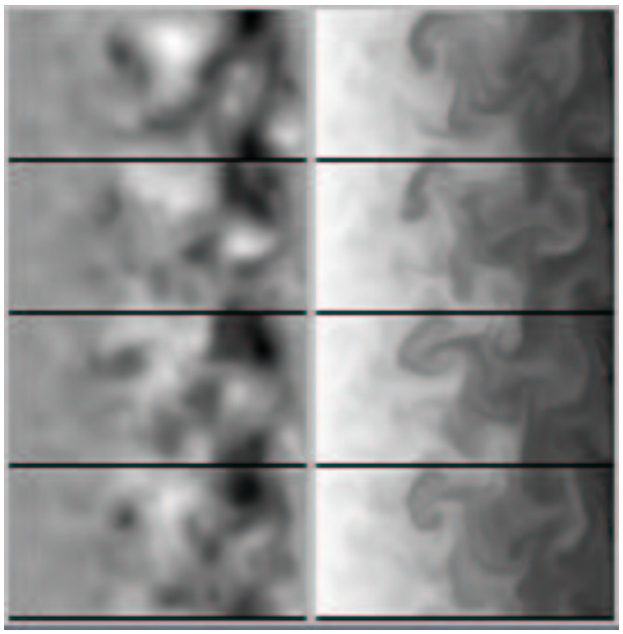

FIG. 4. Nonlocal simulation of ϕ (left) and n (right) at $t = 3000\Omega_i$.

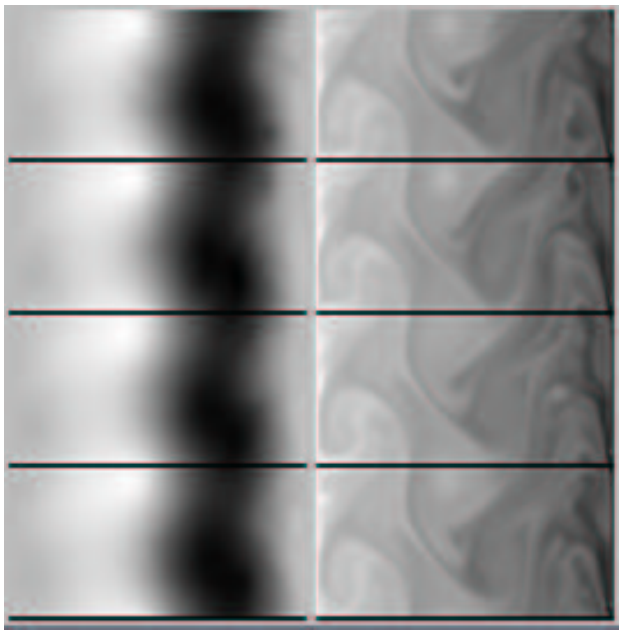

FIG. 5. Nonlocal simulation of ϕ (left) and n (right) at $t = 1.2 \times 10^4\Omega_i$.

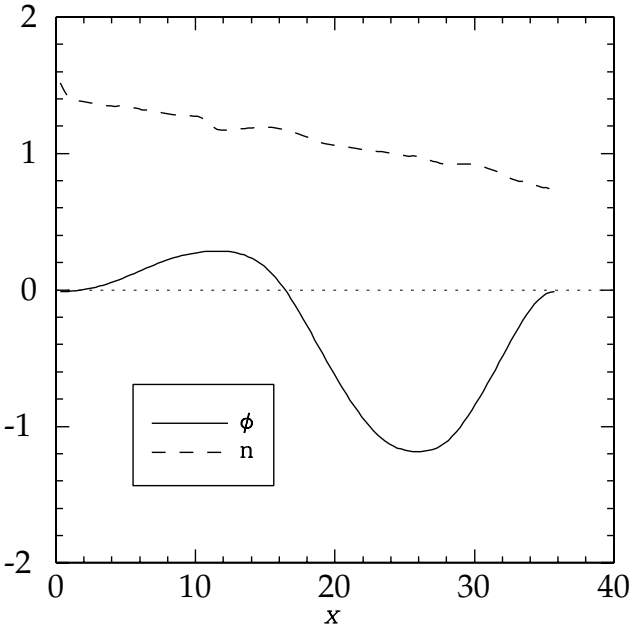

FIG. 6. Density and potential profiles corresponding to Fig. 5.

Recognizing that this savings of roughly an order of magnitude in the toroidal resolution could be transferred to the poloidal direction, we opted to use our efficient multigrid solver to model an entire flux surface and thereby demonstrate the feasibility of nonlocal full flux surface simulations of the (admittedly overly simplistic) Hasegawa–Wakatani equations on a high-performance workstation. We adopted parameters characteristic of the outer edge region of a deuterium L-mode plasma in the ASDEX Upgrade tokamak, for which $R = 165\,\mathrm{cm}$, $a = 50\,\mathrm{cm}$, $B = 2.2\times10^4\,\mathrm{gauss}$, $T_e = 130\,\mathrm{eV}$, $\bar{n} = 6.0\times10^{12}\,\mathrm{cm}^{-3}$, $dn/dx = 2.4\times10^{12}\,\mathrm{cm}^{-4}$, and $q = 3.5$. The effective atomic number Z_{eff} was taken to be 4. These parameters lead to the derived values $\rho_s = 0.075\,\mathrm{cm}$, $L_\parallel = 80.4\,\mathrm{cm}$, and $L_n = 2.5\,\mathrm{cm}$. The boundary values of the density, $n_{\min} = 0.63$ and $n_{\max} = 1.59$, were chosen so that the geometric mean of the normalized density is unity. A $31\times4096\times4$ grid corresponding to a physical domain with $32\rho_s\times4096\rho_s\times13L_\parallel$ ($2.4\,\mathrm{cm}\times307\,\mathrm{cm}\times1045\,\mathrm{cm}$) was adopted.

The simulation was initialized with random values chosen uniformly from the interval $[-10^{-3}, 10^{-3}]$ and started with zero magnetic shear and the dissipation parameters $D_\phi = D_n = 10^{-4}$. To speed up the initial evolution, the coefficient in front of the parallel terms (which restricts the linear time step) was temporarily reduced to 0.001, until the nonlinear phase was reached. The simulation was then run further with the correct parallel coefficient of unity, until the state shown in Fig. 7 was obtained. The long parallel correlation lengths of the turbulence are evident in Fig. 8, where the parameter $\Delta = (z' - z_{\min})/(z_{\max} - z_{\min})$ measures the relative distance along one toroidal circuit of the field line. Although no curvature effects or magnetic field gradients were included in this simulation, we emphasize the doubly periodic boundary conditions by plotting the results in toroidal geometry, using the untwisted coordinates (x, y, z). For presentation purposes, the poloidal direction is compressed by a factor of 4 and the aspect ratio is reduced.

The resulting nonlinear state was then used to initialize a flux-surface run with magnetic shear, taking the shear parameter $\alpha = 2\pi\hat{s}/[q(z_{\max} - z_{\min})]$. Given our normalized parallel simulation length $z_{\max} - z_{\min} = 13$, one computes for the typical shear coefficient $\hat{s} = 1$ that $\alpha = 0.14$. For these parameters Eqs. (13) are linearly stable. However, once in a nonlinear state, the magnetic shear only partially stabilizes the turbulence, as seen in Figs. 9 and 10: the turbulent particle flux is diminished, but not eliminated, upon switching on the magnetic shear at $t = 0$. This illustrates the well-known nonlinear instability mechanism of the Hasegawa–Wakatani equations [12–14]. The results were qualitatively similar whether we held the dissipation parameters fixed or reduced them both by a factor of 1000 to demonstrate that the magnetic-shear damping mechanism is effective even when the dissipation is very weak (as it is physically). We display in Fig. 12 the self-consistent density and potential profiles and observe in Fig. 11 the extremely long parallel correlation lengths in the final turbulent sheared-flow state. On a single-processor 600MHz Digital Alpha PC164LX workstation, the portion of the simulation shown in Fig. 9 required 29 000 adaptive time steps, 300 CPU hours, and 200MB of memory.

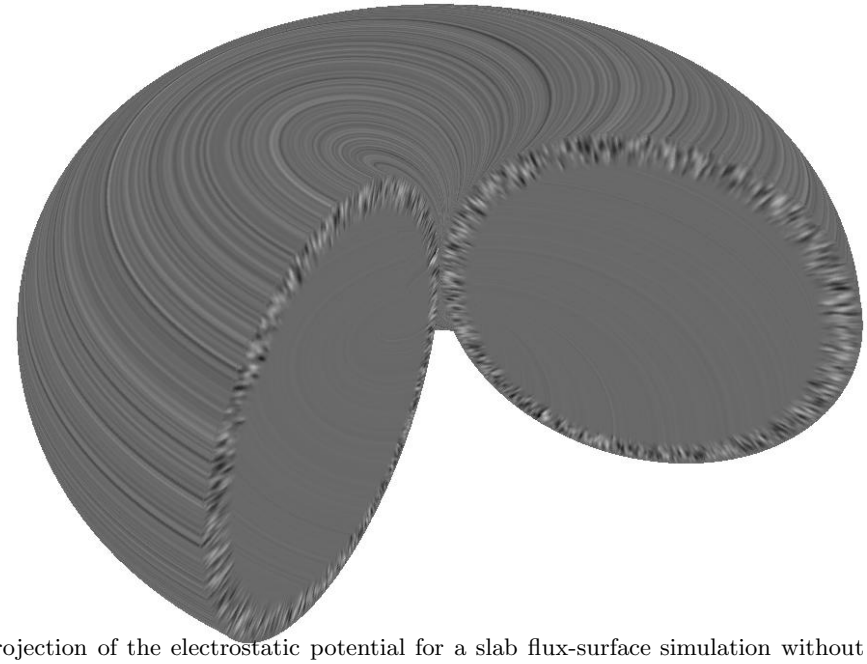

FIG. 7. Toroidal projection of the electrostatic potential for a slab flux-surface simulation without magnetic shear. Black denotes low potential.

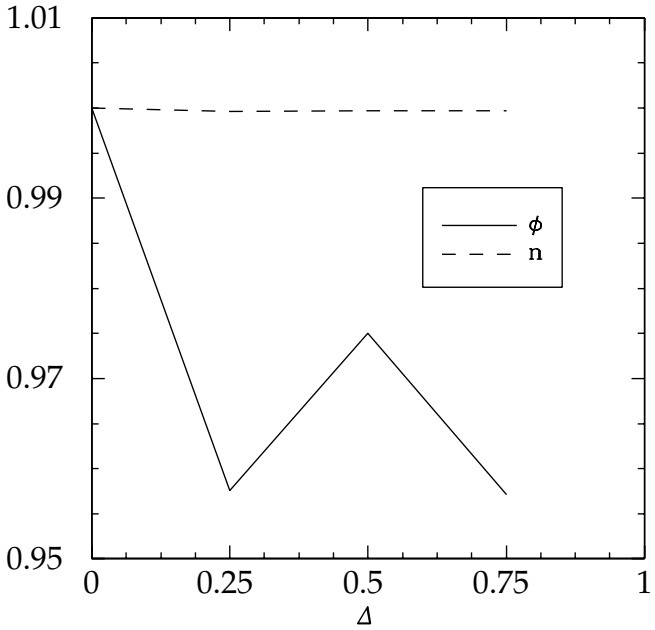

FIG. 8. Density and potential parallel autocorrelation functions corresponding to Fig. 7.

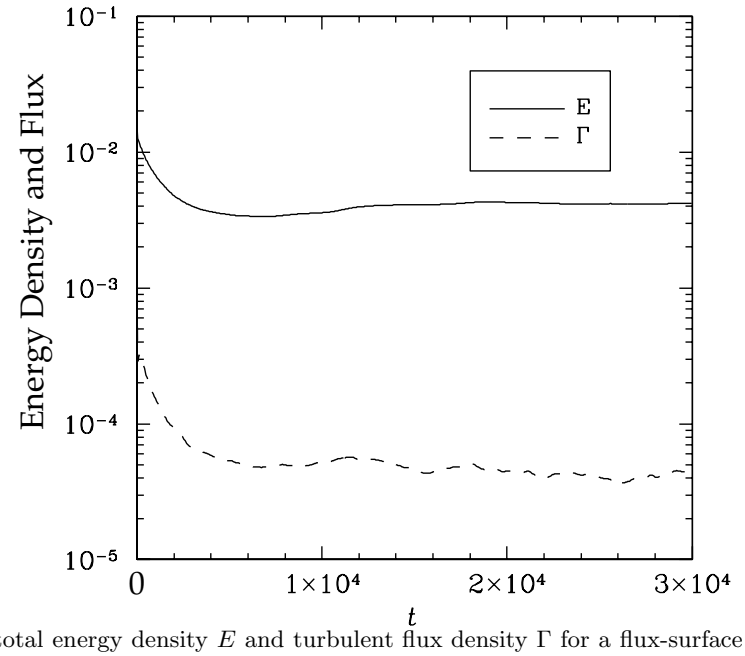

FIG. 9. Evolution of the total energy density E and turbulent flux density Γ for a flux-surface simulation upon turning on the magnetic shear at $t = 0$.

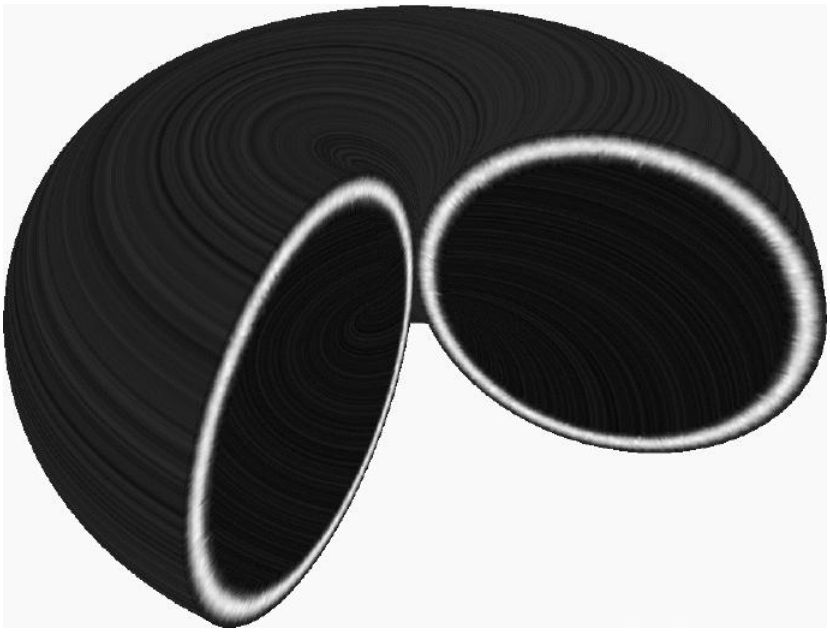

FIG. 10. Toroidal projection of the electrostatic potential for the sheared-slab simulation in Fig. 9 at $t = 3 \times 10^4 \Omega_i$. Black denotes high potential.

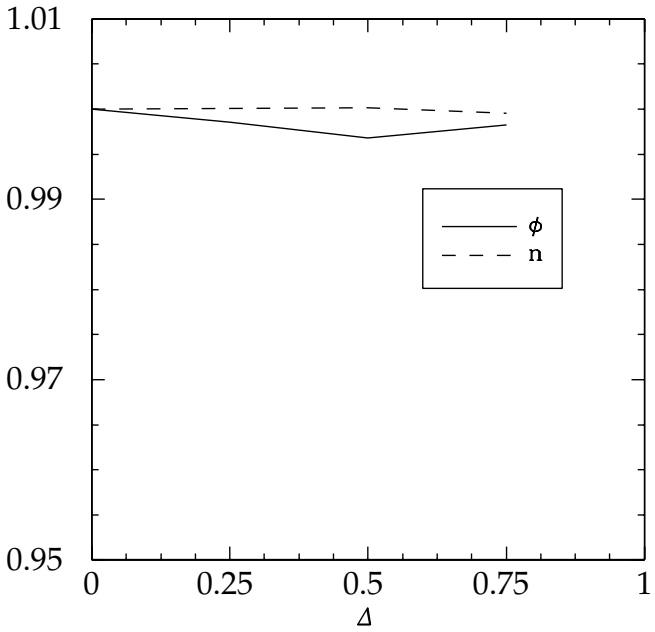

FIG. 11. Density and potential parallel autocorrelation functions corresponding to Fig. 10.

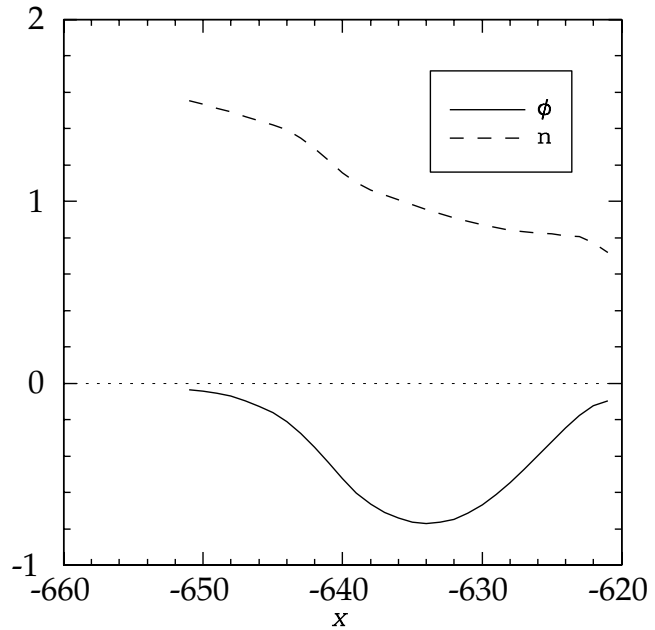

FIG. 12. Density and potential profiles corresponding to Fig. 10.

VI. CONCLUSIONS

An efficient nonlocal anisotropic multigrid solver has been successfully implemented and tested for the Hasegawa–Wakatani Model, using a twisted coordinate system to exploit the extremely long scale lengths along the magnetic field. In addition to illustrating the advantages of multigrid methods for the solution of parabolic partial differential equations, an important purpose of this work was to demonstrate that nonlocal full flux surface calculations of the Hasegawa–Wakatani equations are possible on modern high-performance workstations. In a flux-surface model, the resolution that one saves in following a single toroidal circuit, instead of several connection lengths, can be put into the poloidal direction. The parallel correlation lengths that were obtained in our simulation are evidently much longer than a connection length; however, the inclusion of ballooning physics arising from the neglected magnetic curvature and gradient terms may change this picture.

As a final comment, we note that it appears to be possible to generalize the solver to handle electromagnetic effects. In future work, we would like to explore this further, along with incorporating curvature effects and the complete nonlinear reduced Braginskii equations (*cf.* Ref. [15]), including ion thermal dynamics, into the algorithm. It should be pointed out that realistic electromagnetic computations of tokamak edge turbulence, such as those found in Ref. [16], necessitate the solution of equations much more complicated than Eqs. (13) and require much longer computation times. Moreover, to gain a complete understanding of the underlying physics, many such runs will be required. Since the flux-surface simulation of the Hasegawa–Wakatani equations reported in this work is close to the limit of what is practical on a modern workstation, more realistic simulations of tokamak turbulence will undoubtedly require the use of massively parallel computers.

ACKNOWLEDGMENTS

The authors would like to thank Dr. B. D. Scott for discussions on drift-wave turbulence, boundary conditions, and the multigrid method.

APPENDIX A: MULTIGRID CLASS LIBRARY

A C^{++} class library has been developed around a core recursive multigrid routine to facilitate the development of one-, two-, and three-dimensional multigrid solvers (such as the two- and three-dimensional solvers used in this work). The implementation of a complete multigrid algorithm requires the specification of four principal components: these are the smoother, defect, restriction, and prolongation routines [9].

Default 3^d-point restriction and prolongation algorithms, where d is the dimension of the problem, are defined in the class library. The smoother and defect, on the other hand, must be specified by the user since these are problem dependent. However, some assistance in developing a smoother is provided in the class library. Generally the most effective smoothers are the pointwise or blockwise Gauss–Seidel iterations. Once the Gauss–Seidel iteration for a given operator is coded, the user may select from the class library either a lexicographical, red-black, or line- or surface-zebra ordering of the variables, as described below [9]. A Jacobi iteration is also provided. A variety of predefined boundary conditions are available in each direction and the user may also define more general conditions, such as Eq. (23). The mesh constructed by the initialization routines is based on the type of boundary conditions for each direction (Dirichlet, Neumann, Periodic, Mixed Neumann/Dirichlet), as illustrated in Figure 13.

1. Smoothing iterations

Let us illustrate several of the predefined smoothers of our multigrid class library using the pedagogical example of a centered finite-difference scheme for the two-dimensional isotropic Poisson equation ${\nabla_\perp}^2\phi = f$,

$$\phi_{i+1,j} + \phi_{i,j-1} - 4\phi_{i,j} + \phi_{i,j+1} + \phi_{i-1,j} = f_{i,j}\Delta^2, \tag{A1}$$

where Δ is the grid size, here assumed for simplicity to be the same in both directions.

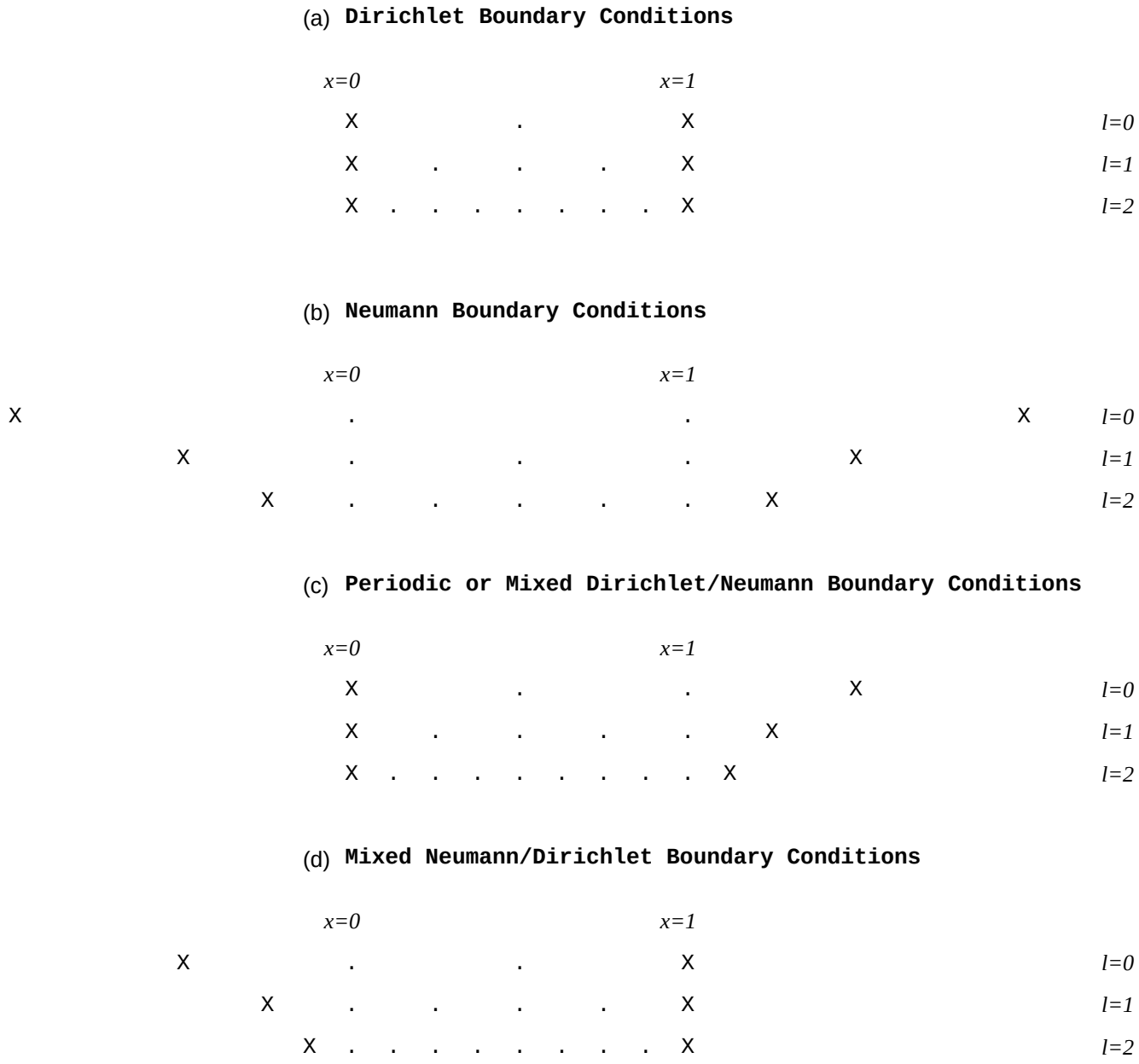

FIG. 13. Grid configurations used for (a) the Dirichlet boundary conditions $\phi(0) = \phi_0$, $\phi(1) = \phi_1$; (b) the Neumann boundary conditions $\phi'(0) = \phi'(1) = 0$; (c) the periodic boundary conditions $\phi(0) = \phi(1)$, $\phi'(0) = \phi'(1)$ or the mixed Dirichlet/Neumann boundary conditions $\phi(0) = \phi_0$, $\phi'(1) = 0$; (d) the mixed Neumann/Dirichlet boundary conditions $\phi'(0) = 0$, $\phi(1) = \phi_1$. A dot denotes a computational grid point and an 'X' designates an inactive (ghost) point used for enforcing the boundary conditions.

a. Jacobi

The Jacobi iteration attempts to obtain a solution to the Poisson equation by relaxing a heat equation. The l-th *damped Jacobi iteration* for Eq. (A1) is given in terms of the previous guess ϕ^{l-1}:

$$\phi^{l}_{i,j} := \phi^{l-1}_{i,j} - \omega \left(\phi^{l-1}_{i-1,j} + \phi^{l-1}_{i,j-1} - 4\phi^{l-1}_{i,j} + \phi^{l-1}_{i,j+1} + \phi^{l-1}_{i+1,j} - f_{i,j}\Delta^2 \right), \tag{A2}$$

where the artificially introduced time step ω controls the rate of numerical relaxation. To obtain an effective smoothing iteration, the value of ω must be chosen carefully, based on an analysis of the eigenvalues of the operator being inverted (for the one-dimensional Poisson equation, the optimal value is one-half of the Courant limit for the corresponding one-dimensional heat equation). This is discussed further in Ref. [9]. However, the Gauss–Seidel iteration discussed next requires no stability parameter and, for elliptic operators, is generally found to be a more effective smoother than the Jacobi iteration.

b. Pointwise Gauss–Seidel

The *pointwise Gauss–Seidel* iteration for Eq. (A1) is given by

$$\phi_{i,j} := \left(\phi_{i-1,j} + \phi_{i,j-1} + \phi_{i,j+1} + \phi_{i+1,j} - f_{i,j}\Delta^2 \right) /4. \tag{A3}$$

It is obtained from Eq. (A1) simply by solving for $\phi_{i,j}$ in terms of $f_{i,j}$ and the other grid values. Unlike for the Jacobi iteration, the ordering of the variables clearly affects the results since the updated values of ϕ are immediately available in subsequent evaluations of the right-hand side.

c. Blockwise Gauss–Seidel

A *blockwise Gauss–Seidel* iteration may also be constructed, by solving Eq. (A1) for an entire block of variables (such as a row or column) using a simpler block-solver (perhaps based on an explicit solution or a lower-dimensional multigrid solver). For example, the *x-line Gauss–Seidel* iteration is defined by

$$\phi_{i-1,j} - 4\phi_{i,j} + \phi_{i+1,j} := f_{i,j}\Delta^2 - \phi_{i,j-1} - \phi_{i,j+1}. \tag{A4}$$

As part of each iteration, an efficient tridiagonal solver can be used to update the values of an entire row simultaneously. Likewise, the *y-line Gauss–Seidel* iteration is defined by

$$\phi_{i,j-1} - 4\phi_{i,j} + \phi_{i,j+1} := f_{i,j}\Delta^2 - \phi_{i-1,j} - \phi_{i+1,j}. \tag{A5}$$

Blockwise solvers like these, especially when combined with zebra-line ordering, are particularly useful for anisotropic problems, where the partial derivatives in one direction dominate. In these situations, the blocks should be chosen to be lines in the dominant direction. Because it solves Eq. (A1) explicitly in the direction of dominance, the resulting blockwise Gauss–Seidel iteration is then able to relax the equation efficiently in the subdominant direction. For example, for a three-dimensional operator possessing extreme anisotropy in the z-direction, one would choose the blocks to be x–y planes, using a zebra ordering of the x–y surfaces.

2. Domain orderings

We now describe a few of the most useful orderings for second-order elliptic problems.

a. Lexicographical

This simplest ordering of the (i, j) values corresponds to the loop

```
for(i=0; i < nx; i++)
    for(j=0; j < ny; j++)
        GaussSeidel(i,j).
```

That is, one repeatedly applies the pointwise Gauss–Seidel iteration (A3), denoted here by `GaussSeidel(i, j)`, scanning through all (i, j) pairs in columns, with the j index increasing most rapidly.

b. Red-black

In this ordering, the points are divided into two sets. One set, consisting of points (i, j) such that $i + j$ is even, is labeled "red" and its complement is labeled "black". Eq. (A3) is first applied to all of the points in the red set:

```
for(i=0; i < nx; i += 2)
    for(j=0; j < ny; j += 2)
        GaussSeidel(i,j)
for(i=1; i < nx; i += 2)
    for(j=1; j < ny; j += 2)
        GaussSeidel(i,j).
```

On a second pass, the iteration is then applied to each of the points in the black set:

```
for(i=0; i < nx; i += 2)
    for(j=1; j < ny; j += 2)
        GaussSeidel(i,j)
for(i=1; i < nx; i += 2)
    for(j=0; j < ny; j += 2)
        GaussSeidel(i,j).
```

c. Line-zebra

The *x-line zebra* ordering is defined by

```
for(j=0; j < ny; j += 2)
    XGaussSeidel(j)
for(j=1; j < ny; j += 2)
    XGaussSeidel(j),
```

where `XGaussSeidel(j)` is the x-line Gauss–Seidel iteration (A4). Similarly, the *y-line zebra* ordering is defined by

```
for(i=0; i < nx; i += 2)
    YGaussSeidel(i)
for(i=1; i < nx; i += 2)
    YGaussSeidel(i),
```

where `YGaussSeidel(i)` is the y-line Gauss–Seidel iteration (A5).

d. Surface-zebra

For three-dimensional problems, one can define surface-zebra orderings such as the *xy-surface zebra* ordering

```
for(k=0; k < nz; k += 2)
    XYGaussSeidel(k)
for(k=1; k < nz; k += 2)
    XYGaussSeidel(k),
```

where `XYGaussSeidel(k)` is an xy-surface Gauss–Seidel iteration, in which the blocks are chosen to be x–y planes. This ordering is useful for problems possessing extreme anisotropy in the z-direction. The solution on each x–y plane would normally be obtained with a secondary two-dimensional multigrid solver.

3. Boundary conditions

We now describe the predefined boundary conditions corresponding to the one-dimensional grids depicted in Fig. 13. Analogous routines, which may be combined to yield a wide variety of boundary conditions, are available in higher dimensions.

a. Dirichlet conditions

The multigrid package assumes that the desired Dirichlet boundary conditions are initially applied to the approximate solution given by the user. Therefore, no additional code needs to be executed to enforce Dirichlet boundary conditions in the multigrid algorithm.

b. Neumann conditions

The following boundary conditions are applied to the solution and defect:

$$u_0 = u_2, \tag{A6}$$

$$u_{n+1} = u_{n-1}. \tag{A7}$$

Here n is the number of active computational points; including the two boundary points there are a total of $n + 2$ points.

c. Mixed Dirichlet/Neumann conditions

There are two situations, one with Neumann boundary conditions at $x_{\min}$,

$$u_0 = u_2, \tag{A8}$$

and one with Neumann boundary conditions at $x_{\max}$,

$$u_{n+1} = u_{n-1}. \tag{A9}$$

d. Periodic conditions

Periodic boundary conditions are implemented with the following assignments:

$$u_0 = u_n \tag{A10}$$

$$u_{n+1} = u_1. \tag{A11}$$

---

[1] A. Hasegawa and M. Wakatani, Plasma Edge Turbulence, *Phys. Rev. Lett.* **50**, 682 (1983).
[2] W. Horton, in *Handbook of Plasma Physics*, edited by M. N. Rosenbluth and R. Z. Sagdeev (North-Holland, Amsterdam, 1984), Vol. 2: *Basic Plasma Physics II*, edited by A. A. Galeev and R. N. Sudan, Chap. 6.4, p. 383.
[3] P. N. Guzdar, J. F. Drake, D. McCarthy, A. B. Hassam, and C. S. Liu, Three-dimensional fluid simulations of the nonlinear drift-resistive ballooning modes in tokamak edge plasmas, *Phys. Fluids B* **5**, 3712 (1993).
[4] A. Zeiler, D. Biskamp, J. Drake, and P. Guzdar, Three-dimensional fluid simulations of tokamak edge turbulence, *Phys. Plasmas* **3**, 2951 (1996).
[5] R. L. Dewar and A. H. Glasser, Ballooning mode spectrum in general toroidal systems, *Phys. Fluids* **26**, 3038 (1983).
[6] M. A. Beer, S. C. Cowley, and G. W. Hammett, Field-aligned coordinates for nonlinear simulations of tokamak turbulence, *Phys. Plasmas* **2**, 2687 (1995).
[7] B. Scott, Three-dimensional computation of collisional drift wave turbulence and transport in tokamak geometry, *Plasma Phys. Control. Fusion* **39**, 471 (1997).
[8] B. Scott, Global consistency for thin flux tube treatments of toroidal geometry, *Phys. Plasmas* **5**, 2334 (1998).

[9] W. Hackbusch, *Multi-Grid Methods and Applications*, Series in Computational Mathematics (Springer, New York, 1985).
[10] W. L. Briggs, *A multigrid tutorial* (SIAM, Philadelphia, 1987).
[11] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes, The Art of Scientific Computing*, 2nd ed. (Cambridge Univ. Press, Cambridge, 1992).
[12] J. F. Drake, A. Zeiler, and D. Biskamp, Nonlinear Self-Sustained Drift-Wave Turbulence, *Phys. Rev. Lett.* **75**, 4222 (1995).
[13] B. D. Scott, Self-sustained collisional drift-wave turbulence in a sheared field, *Phys. Rev. Lett.* **65**, 3289 (1990).
[14] D. Biskamp and A. Zeiler, Nonlinear Instability Mechanism in 3D Collisional Drift-Wave Turbulence, *Phys. Rev. Lett.* **74**, 706 (1995).
[15] A. Zeiler, J. F. Drake, and B. Rogers, Nonlinear reduced Braginskii equations with ion thermal dynamics in toroidal plasma, *Phys. Plasmas* **4**, 2134 (1997).
[16] B. N. Rogers, J. F. Drake, and A. Zeiler, Phase Space of Tokamak Edge Turbulence, the L-H Transition, and the Formation of the Edge Pedestal, *Phys. Rev. Lett.* **91**, 4396 (1998).