Eigenvalue solver for fluid and kinetic plasma models in arbitrary magnetic topology

D. A. Baver and J. R. Myra

Lodestar Research Corporation, Boulder, CO, USA

M. V. Umansky

Lawrence Livermore National Laboratory, Livermore, CA, USA

September 2014

submitted to *Computer Physics Communications*

DOE-ER/SC0006562-2 LRC-14-157

LODESTAR RESEARCH CORPORATION

2400 Central Avenue

Boulder, Colorado 80301

Eigenvalue solver for fluid and kinetic plasma models in arbitrary magnetic topology

D. A. Baver and J. R. Myra

Lodestar Research Corporation, Boulder Colorado 80301

M. V. Umansky

Lawrence Livermore National Laboratory

ArbiTER (Arbitrary Topology Equation Reader) is a new code for solving linear eigenvalue problems arising from a broad range of physics and geometry models. The primary application area envisioned is boundary plasma physics in magnetic confinement devices; however ArbiTER should be applicable to other science and engineering fields as well. The code permits a variable numbers of dimensions, making possible application to both fluid and kinetic models. The use of specialized equation and topology parsers permits a high degree of flexibility in specifying the physics and geometry.

I. INTRODUCTION

Most modern computational efforts for simulating the edge region of fusion plasmas employ time advancement to capture the nonlinear and turbulent evolution of the particles and fields. These simulation codes employ either fluid models [1]-[2] or full kinetic simulation using particle-in-cell (PIC) [3] or Vlasov fluid approaches [4] all of which are implemented in the time domain. The plasma simulation community needs such tools for theoretical analysis, numerical experimentation, experimental modeling, and even for the hardware component design.

On the other hand, there is a small but significant class of important edge plasma problems which are amenable to linear and/or quasilinear analysis. These include source-driven problems and eigenvalue problems such as those arising in the computation of the growth rates of linear plasma instabilities.

Time-evolution codes can be used for treating such problems, however there are advantages in using a non time-evolution approach. Indeed, consider a general time-evolution equation

$$d\vec{f}/dt = \vec{F}(\vec{f}, x, t) \tag{1}$$

In the case where the problem is linear, or can be linearized, the right-hand side can be represented as a matrix M, i.e.,

$$d\vec{f}/dt = \hat{M}(x, t)\vec{f} \tag{2}$$

Furthermore we are assuming M(x,t)=M(x), with no explicit time dependence. In this case, for calculation of the linear instabilities in this system, one can calculate the eigenvalues and eigenmodes of the matrix M by the methods of linear algebra,

$$\hat{M}\vec{f} = -\omega^2 \vec{f} \tag{3}$$

This approach has advantages over solving the problem by time-evolution: it is typically more efficient in terms of CPU-hours needed to solve a problem of given resolution, and it is capable of finding subdominant modes, i.e. modes with less than the maximum growth rate. Such modes, even if stable, can lend useful insights into the underlying physics. [5]

Similarly, one can consider the problem of linear response:

$$d\vec{f}/dt = \hat{M}(x)\vec{f} + \vec{g}(x,t) \tag{4}$$

where $\vec{g}(x,t)$ is the forcing term.

Instead of solving by time-evolution, one can take advantage of the linearity and assume a single frequency time-dependence, $\vec{g}(x,t) = \vec{g}(x)\exp(-i\omega t)$. Assuming no secular terms and assuming the homogeneous part of the solution decays in time, one can solve for the asymptotic stationary solution by solving the linear system

$$(i\omega\hat{I} + \hat{M}(x))\vec{f} = -\vec{g}(x). \tag{5}$$

Finally, linear codes can also play a role in understanding some classes of nonlinear problems which are amenable to quasilinear analysis. The appropriate biquadratic forms can readily be calculated once the linear eigenfunctions are known.

When a linear or quasilinear treatment is appropriate there are multiple advantages to using a code which exploits this feature. For the same problem, a linear code can typically run much faster, with much better resolution. Linear algorithms tend to be very robust computationally, and numerical convergence is usually more straightforward to diagnose and achieve. For these reasons, linear codes are very well suited to parameter space studies, and can promise a high degree of confidence in their results. The latter attribute makes them especially important for use as verification tools [6]-[7] in comparative studies with their nonlinear time-domain counterparts. Quasilinear calculations, performed as a post-processing step, provide fast and robust results which can provide insight into some processes, such as induced forces and transport fluxes.

The Arbitrary Topology Equation Reader (ArbiTER) code is a general tool for discretizing and solving linear partial differential equations. It is an extension of the 2DX code [8], a code which was developed to solve a general class of fluid-based eigenvalue problems in the edge and scrape-off layer region of a diverted tokamak. ArbiTER inherits much of its code structure from 2DX, as well as the essential characteristics of its equation parser. What the ArbiTER code adds is a topology parser. This permits radical changes in the connectivity of grid points without any fundamental change in the code itself. ArbiTER can

emulate 2DX, but in addition it capabilities extend far beyond those of 2DX. The number of dimensions in a problem can be varied in ArbiTER, such that the same basic code can be used to solve both fluid and kinetic models. It is capable of arbitrary reconfiguration of topological regions, allowing simulation of advanced divertor configurations such as snowflakes [9]. It is even capable of performing calculations using unstructured grids, although this feature has not been thoroughly applied in plasma physics applications to date.

As with 2DX, the flexibility and modularity of ArbiTER have considerable advantages from the standpoint of code verification. By splitting the modeling of each problem into two parts, the equation and topology input files and the ArbiTER source code, it allows each part to be independently verified by different means; the source code can be verified on other models because the same basic routines are used in many different types of models, whereas the equation input file (or "structure" file) describing the physics model can be converted into algebraic form for ease of inspection. This modularity has further advantages from the standpoint of basic physics studies. By separating equation (structure) and topology features into different inputs, it is possible to mix and match equations and topologies. Thus, a model that has already been demonstrated on a simple topology can operate reliably on an entirely new topology. Likewise, once a topology has been tested, it is relatively easy to change the model equations being solved.

II. PROGRAM STRUCTURE

The ArbiTER code consists of two main parts, as well as a number of tools required to set up its input files and process its output files. The relation between these parts is shown in Fig. 1. Of these, the grid generation tools and the data analysis tools are currently implemented as Mathematica notebooks, although Python scripts for grid generation are also available. Conversion between the editable and machine-readable versions of the structure and topology file are done using Python scripts. The Field/Domain variant referred to in Fig. 1 is an alternate build of the ArbiTER kernel. This program permits grid files to be generated via an iterative process, in which information stored in the topology file is used to generate an intermediate file that is then in turn used to aid in formatting data in the grid file.

The principal parts of the ArbiTER code are the ArbiTER kernel and the eigenvalue solver. The ArbiTER kernel generates a pair of sparse matrices in coordinate list format (sometimes referred to as COO format), that is, a list of entries each containing the value, row and column of each non-zero entry in the matrix. These matrices describe a generalized eigenvalue problem,

$$Ax = \lambda Bx \tag{6}$$

The pair of matrices is passed to an eigenvalue solver, which returns the eigenvalues λ and eigenvectors x. The eigenvalue solver used by ArbiTER is SLEPc [14]. In addition, there exist variants of the ArbiTER code that are

discussed in greater detail in Sec. C. One of these uses the same matrices A and B generated by the ArbiTER kernel but solves them as ordinary matrix equations,

$$Ax = Bb \tag{7}$$

Problems of this "source-driven" type are solved using PETSc [15], which is already included in the process of installing SLEPc. In either case, a variety of command-line options are available with the respective solver packages. In the case of SLEPc, the use of these options is discussed in the reference on the 2DX code [8].

The major distinguishing features of the ArbiTER code are related to how it builds the matrices A and B that define the eigenvalue problem. Like its predecessor 2DX, ArbiTER uses elementary matrix operations to build these matrices from simpler matrices. Unlike 2DX, rather than rely on built-in elementary operators as basic building blocks, ArbiTER allows the user to define their own elementary operators using a topology parser. This allows the user to define the relations in coordinate space between nodes in their computational domain, and from that information define differential operators that take the connectivity of those nodes into account. This ability to define both computational domains and differential operators gives the code its immense flexibility, making it suitable for both higher-dimensional models such as kinetic models as well as problems involving unstructured grids.

A. Equation language

The equation language is a file input format for defining sets of partial differential equations to discretize and solve. The resulting file, henceforth referred to as the structure file, contains all of the information the ArbiTER code needs to convert profile functions (i.e. input data such as temperature, density, magnetic geometry, etc. needed to define the matrix elements) into sparse matrices to solve. The equation language used in ArbiTER is based on that used in 2DX. This format is discussed in more detail in the paper on 2DX [8]. For clarity some general characteristics of this feature are reviewed here.

The equation language comes in two forms, a machine-readable form used by the ArbiTER code, and a human-readable form for editing. The human-readable form of the ArbiTER equation language is nearly reverse-compatible with 2DX, whereas the machine-readable form is not. The input file contains a number of sections, of which three are of special importance: the input language, the formula language, and the element language.

The input language is used to interpret the main data input file, known as the grid file. Each entry in the input language declares a variable name and assigns it a variable type (i.e. integer, real, function, etc.). In the case of functions the variable is assigned a computational domain, thus defining which entries in the function are assigned to which coordinate positions. The code then searches the input file (either ASCII or HDF5) for the appropriate variable name and loads the associated data block into an appropriate structure in memory.

The formula language constructs the partial differential equation to be solved

from some combination of functions, differential operators, and constants. These elements may be input as profile functions, constructed as elementary operators using the topology language, or built from simpler elements using the element language.

The element language builds elements (functions, constants, or operators) from simpler elements. This is particularly useful when using specialized coordinate systems, such as field-line following coordinates, as the definition of simple gradient operators depends on functions defining magnetic geometry and is quite complicated. Unlike 2DX, the ArbiTER element language can output constants, which is useful for applying physical formulas to the data.

B. Topology language

The main feature that distinguishes ArbiTER from 2DX is the topology language. This is an input file format for defining all features related to the spatial connectivity of grid points. This is primarily concerned with defining topological regions in the computational domain, but additionally is concerned with defining elementary differential operators used by the equation language. The capabilities of this language are fairly general, and among other things allow the ArbiTER code to emulate the capabilities of 2DX. The topology used to emulate 2DX is shown in Fig. 2, namely single-null x-point geometry with three topological regions: a closed flux surface region, the main scrape-off layer and the private flux region.

The topology language consists of a list of definitions for four different types

of objects: blocks, linkages, domains, and operators. Blocks are simple Cartesian sub-domains, used as a building block for domains. Linkages are sparse matrices that describe how to connect two blocks or domains. Domains are sets of grid points that can be assigned to a profile (input) function or a dependent variable. Operators are sparse matrices that describe how grid points are related to one another, e.g. for the purpose of calculating derivatives. In addition, the topology language permits renumbering of domains to ensure that grid points are sequenced in an intuitive order, and the topology language file contains an integer language (integer analog of the element language) to aid in calculating the sizes of blocks and offsets of linkages.

Much of the diversity of the topology language stems from the wide variety of linkage and domain subtypes. Linkages can simply tie together blocks, or they can be multiplied by complex functions to produce phase-shift periodic boundary conditions (relevant to periodicity in toroidal domains described by field-line following coordinates), or they can even be defined by using an integer array to specify individual grid cells to be connected. Domains likewise can be simply assembled from blocks and linkages, or they can be constructed from existing domains in a number of different ways, such as convolution (combining two domains to create an outer product space) or indentation (removing cells from selected boundaries, relevant to creating staggered grids). These features provide significant advantages for kinetic models, in which operators must be defined that link domains of different dimensionalities.

C. Variant builds

While the core ArbiTER code was designed to solve eigenvalue problems, a number of variant builds have been developed for more specialized purposes. Some are intended as tools for debugging equation and topology input files, some are intended as tools for extracting geometric and topological information for purposes of formatting profile function inputs, while some introduce entirely new capabilities to the code.

The full list of variant builds is shown in Table 1. In addition, many of the variants on this list can be built with or without HDF5 file capability. This allows the program to be compiled on machines that do not have HDF5 installed. Some variants do not use SLEPc, and can therefore be compiled on machines that do not have SLEPc installed. In this case, matrices can be generated for debugging or for use in external solvers.

Some of these variants merit special attention. In particular, the field/domain variant (arbiterfd) can be used in the process of gridfile generation as shown in Fig 1. In this case, the field/domain variant is run using the same topology file as the intended model equations, but a simplified structure file. The purpose of this is to generate a file containing a list of nodes and their coordinate positions for each domain. This is important because ArbiTER inputs profile functions as 1-D arrays. In order to input arrays with a larger numbers of dimensions, data values must be placed in a specific sequence, which is determined when the domain is generated internally. If the domain is particularly simple, external routines can be written to sequence the data correctly. However, for more

complicated domains, this means the topology must be coded twice: once in the topology file and once in the grid setup routine. By using the field/domain variant to calculate the correct data sequence, one only needs to generate the topology once, and this same topology is used both to create the grid file and to generate the full eigenvalue problem. In addition to creating domains, the field/domain variant can also be used to process constants (for instance to calculate basic plasma parameters) and to process profile functions. The latter case assumes that data sequenced according to the correct input topology is already available from somewhere, but this can still reduce the complexity of grid file setup in some cases, particularly if the profile functions are to be interpolated or truncated after being calculated.

III. VERIFICATION AND TEST CASES

A. Resistive ballooning in snowflake geometry

One of the most basic advantages of the ArbiTER code over its predecessor 2DX is the ability to change topologies easily. This is particularly important in models that use field-line following coordinates, as any change in the magnetic topology will correspondingly change the structure of the computational grid in non-trivial ways. As it turns out, field-line following coordinates are commonly used in both fluid and kinetic models for studying plasma edge instabilities, since the large anisotropies found in magnetized plasmas tend to result in eigenmodes that are highly localized in the radial direction but extended along field lines.

A particularly relevant demonstration of this capability is the application of a simple plasma model in a snowflake divertor [9]. The snowflake divertor is based on a higher-order null than an x-point divertor, resulting in weaker poloidal fields near the null. In practice, a snowflake divertor is better described as containing two separate x-points in close proximity to one another. Depending on the relative positions of these x-points, the snowflake may be described as snowflake-plus or snowflake-minus. The case used in this study is of the snowflake-minus variety.

For purposes of simplicity and ease of comparison to preexisting benchmark cases, a resistive ballooning model [10]-[12] was used for this test. The model equations for this are as follows:

$$\gamma \nabla_{\perp}^2 \delta\Phi = +\frac{2B}{n} C_r \delta p - \frac{B^2}{n} \partial_{\parallel} \nabla_{\perp}^2 \delta A \tag{8}$$

$$\gamma \delta n = -\delta v_E \cdot \nabla n \tag{9}$$

$$-\gamma \nabla_{\perp}^2 \delta A = \nu_e \nabla_{\perp}^2 \delta A - \mu n \nabla_{\parallel} \delta\Phi \tag{10}$$

where:

$$\delta p = (T_e + T_i)\delta n \tag{11}$$

$$C_r = \mathbf{b} \times \kappa \cdot \nabla = -\kappa_g R B_p \partial_x + i(\kappa_n k_z - \kappa_g k_\psi) \tag{12}$$

$$\nabla_\perp^2 = -k_z^2 - \jmath B(k_\psi - i\partial_x R B_p)(1/\jmath B)(k_\psi - i R B_p \partial_x) \tag{13}$$

$$\partial_\parallel Q = B\nabla_\parallel (Q/B) \tag{14}$$

$$\nabla_\parallel = \jmath \partial_y \tag{15}$$

$$\delta v_E \cdot \nabla Q = -i\frac{k_z (R B_p \partial_x Q)}{B}\delta\Phi \tag{16}$$

$$\nu_e = .51 \nu_r n / T_e^{3/2} \tag{17}$$

Here the growth rate γ is the eigenvalue for a three-field model coupling perturbed electrostatic potential $\delta\Phi$, density δn, and parallel vector potential δA. C_r is the curvature operator, κ_n and κ_g are the normal and geodesic curvatures, respectively, ν_e is the electron-ion collision frequency, μ is the mass ratio, $\jmath$ is the Jacobian, and in Eqs. 14 and 16 Q is an arbitrary profile.

These model equations were applied to a snowflake profile based on DIII-D data. Ad-hoc illustrative density and temperature profiles were generated during pre-processing. The test cases used a flat temperature profile with density defined by a tanh function with respect to poloidal flux. By adjusting the position of the inflexion point of the tanh function, it was possible to control the location of the most unstable eigenmode. This in turn permitted study of the effect of magnetic topology on the structure and growth rate of a resistive ballooning mode.

The results of this are shown in Fig. 3. As the peak gradient region is

moved outward, the eigenmode shifts outward accordingly. In the case of the middle plot, one can see an eigenmode localizing between the two x-points. The second from the right shows an eigenmode that is localized near an x-point, and shows forking with the eigenmode having a significant amplitude in two different divertor legs.

B. Kinetic resistive ballooning in x-point geometry

Because the ArbiTER topology language treats dimensionality as a free parameter to be defined as a computational domain is defined, it is possible to apply the code to higher-dimensional gyrokinetic models [13]. However, such models are typically quite complicated, and hence not natural first choices for developing and benchmarking a new code. Also, kinetic models may develop numerical issues that are not typically found in corresponding fluid models. For this reason, it was decided to test the kinetic capabilities of the ArbiTER code on a simple kinetic model. For this purpose, the model chosen was a resistive ballooning model with kinetic parallel electrons.

The model equations in this case are:

$$\gamma \nabla_\perp^2 \delta\phi = \frac{2B}{n} C_r T \delta n + \frac{B^2}{n} \int_v dv v \partial_\parallel \delta f \tag{18}$$

$$\gamma \delta n = -\delta v_E \cdot \nabla n \tag{19}$$

$$\gamma \delta f = -\mu \nabla_\parallel \delta\phi f_0' - v \partial_\parallel \delta f + \nu \mu T_e \partial_v \delta f \tag{20}$$

Again the growth or decay rate γ is the sought-after eigenvalue, f_0 and δf

are the equilibrium and perturbed distribution functions, which depend on the spatial coordinate x and the velocity coordinate v.

In this equation set, there are two computational domains: a $nx \times ny$ domain for real space, and a $nx \times ny \times nv$ domain for phase space. Here f is defined as residing in phase space, whereas ϕ is defined as residing in real space. This takes advantage of features in the ArbiTER topology language: not only is the dimensionality of a computational grid selectable, but computational grids with different dimensionalities can coexist in the same model.

The results of this case are shown in Fig. 4. In order to verify correct construction of the structure and topology files, a test case was run in which the velocity advection terms in the model equation were turned off. This results in a model that is mathematically identical to a fluid model. Results from that are listed as ArbiTER fluid. The results from the fluid and kinetic (full model) cases were then compared to results from a spectral calculation performed using Mathematica, in one of which cases kinetic effects were modeled using the plasma response function. As expected, for the fluid case there is close agreement between the spectral model and the ArbiTER result. In the kinetic case, the ArbiTER code underestimates the growth rate by a few percent. This is adequate agreement given the differences between the treatment of collisions in Eq. 20 and the spectral code which employed a Krook-model collision operator.

C. Heat diffusion using first-order finite element analysis

A significant feature of the ArbiTER code is the ability to input connectivity matrices as integer input from the grid file. The intended use of this feature is to permit the definition of unstructured grids for finite element analysis. While this feature has so far not been applied in the context of edge plasma physics, it is nonetheless an interesting feature worthy of a demonstration case.

The computational domains for this problem begin with two domains, one for nodes and one for cells. The domain for cells is convolved with a domain containing three elements to create a $ncells \times 3$ domain and a $ncells \times 3 \times 3$ domain. A linkage is constructed from integer data in the grid file linking nodes to the $ncells \times 3$ domain. This defines which nodes are included in which cells. Once this is done, differential operators are defined on the $ncells \times 3 \times 3$ domain. By multiplying these operators by operators describing linkages between the $ncells \times 3 \times 3$ and $ncells \times 3$ domains and between the $ncells \times 3$ and $nnodes$ domains, one can map these operators back to node space. Once this is done, the equations can then be defined with dynamical variables existing on the node space.

Once differential operators have been defined on the finite element grid, one can proceed with construction of a suitable grid file. This was done by using a Python script to convert output from a Delaunay triangulation program called Triangle [16]. The resulting final grid contains 3007 nodes. Zero-derivative boundary conditions were used in this test case.

Results from this test are shown in Fig. 5. The fourth eigenmode was shown

for clarity, as lower eigenmodes displayed few distinctive features. The resulting solution is smooth given that only a first order method was used, and has the expected form for a solution to the model equations.

D. Test of source-driven capability on a magnetic dipole

To test the source-driven capability of the ArbiTER code, it was used to calculate the fields of a point magnetic dipole inside a superconducting cylinder. This problem was chosen because an analytic solution had already been calculated for another purpose, otherwise the choice of test problem is largely arbitrary.

In this problem, we place a magnetic dipole along the axis of a cylinder, oriented perpendicular to that axis. We then define a potential, much as one would in the case of an electric dipole; the difference is merely in the boundary conditions, which are zero derivative with respect to this potential as opposed to zero value as they would be in the case of an electric dipole. It can be shown that variation in this potential is purely sinusoidal in the azimuthal direction, corresponding to an $n = 1$ mode. Given this, we can then write down the differential equation for this potential:

$$\frac{\partial^2}{\partial r^2}\psi + \frac{1}{r}\frac{\partial}{\partial r}\psi - \frac{1}{r^2}\psi + \frac{\partial^2}{\partial z^2}\psi = S \tag{21}$$

This problem can be solved in a semi-analytic manner by separation of variables. This results in a series expansion in terms of cosines or hyperbolic cosines and Bessel functions. This solution is described in more detail in the appendix.

Solution of this problem in ArbiTER is relatively straightforward. The formula in Eq. 21 is written in equation language, with the differential operators $\partial^2/\partial r^2$, $\partial/\partial r$, and $\partial^2/\partial z^2$ defined in such a way as to implement zero-derivative boundary conditions on all outer boundaries and zero-value boundary conditions on the central axis. Radius is input as a 1D profile function, which is then distributed across the entire domain. The source term is a product of two 1D profile functions (one for r and one for z) which are distributed across the domain before being multiplied.

The analytic and numerical solutions are compared in Fig. 6. In this case, the analytic solution was a superposition of 30 modes, and the numerical solution was calculated at a grid resolution of 100 radial points and 101 axial points. A 1% discrepancy between the solutions exists for the data shown, however, this discrepancy was found to scale with the convergence conditions specified as command-line options to the matrix solver. It is therefore presumed that this discrepancy can be made asymptotically small.

E. Extended domain method in x-point geometry

In the 2DX emulation topology (i.e. for a single-null x-point geometry plasma), there is a branch cut next to the x-point across which a phase-shift periodic boundary condition is applied, i.e. the eigenfunction on one side of the branch cut is matched to that on the other side times a phase factor which accounts for the toroidal periodicity of the modes [8]. In this section we present an alternative to this technique. The motivations for this and the situations where

such a technique may have an advantage are a topic for future discussion. For purposes of this paper, it is sufficient to discuss the methods by which such a technique are implemented in the ArbiTER code.

In the extended domain method, each field line in a tokamak plasma is extended for multiple poloidal transits. With each successive poloidal transit, the phase shift between adjacent flux surfaces accumulates, and eventually has a suppressive effect on the instability of interest. The effect is related to the integrated magnetic shear. This results in an eigenmode that is roughly centered in a domain constituting multiple poloidal transits, with amplitude decaying exponentially to either side. If the amplitude decays to a sufficient degree before reaching the boundaries of the extended domain, the boundary conditions at the ends of the domain have little effect on growth rates or mode structure, so the introduction of such an artificial boundary does not affect the validity of the computational result. The extended domain method is commonly applied in codes which operate on closed flux surfaces, and is closely relate to the well-known ballooning formalism [17] However, the present application to complex separatrix-spanning topologies is, to the best of our knowledge, original.

This type of method can be implemented easily in ArbiTER by using a third dimension to represent the number of transits in the extended domain. By introducing an offset in the linkage across the branch cut, one end of the edge in each layer connects it to the next layer. A similar domain lacking the extra dimension is constructed to load in profile functions. An operator linking the reduced-dimensionality domain to the extended domain allows these profile

functions to be projected onto the extended domain in a convenient manner. The layout of topological subdomains used in this method is shown in Fig. 7.

Results are shown in Figs. 8 and 9. Fig. 8 shows the sum of the amplitudes for all three periods of the extended domain, superimposed on a single period of the domain. The resulting eigenmode is localized near the separatrix. In Fig. 9, the amplitude in the full extended domain is shown. In this plot, we can see the decay of the amplitude as one moves away from the center period. We can also see that the amplitude outside the separatrix is largely localized to the center period.

IV. SUMMARY

A new eigenvalue solver, the ArbiTER code, has been developed. It is capable of solving linear partial differential equations (including certain classes of integral operators) in arbitrary topology and in an arbitrary number of dimensions. While its development is primarily motivated by problems in edge plasma physics, its capabilities are not limited to such applications.

The ArbiTER code has been tested in a variety of cases that demonstrate its capabilities. These include kinetic plasma models, fluid models in snowflake or extended domain topology, and simple finite element analysis. For application to large scale, high dimensionality problems, the ArbiTER code implements the parallel version of SLEPc [14] with MPI. Ongoing and future work will explore advanced gyrokinetic models and the resulting computation requirements and

parallel scaling.

The ArbiTER code shows tremendous potential both as a benchmarking aid and in primary physics studies. Its flexibility allows it to be applied in a wide variety of situations, and can thus be compared against nearly any code for which a relevant linear regime exists. Moreover, it can readily assimilate a variety of mathematical and numerical techniques, allowing essentially the same model equations to be solved by different methods, for instance comparing extended domain techniques to phase-shift periodic boundary conditions. Such capabilities make the ArbiTER code a valuable tool for exploratory research in plasma physics.

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy Office of Science, Office of Fusion Energy Sciences under Award Number DE-SC0006562.

Appendix: analytic solution for dipole problem

Eq. 21 in Sec. D. can be solved analytically by expanding in a series. This can be done in two different ways. The first is to expand as a series of Bessel functions in r, then solve the resulting differential equation in z for each component. This results in the equation:

$$\psi_1(r,z) = S\sum_{m=1}^{\infty} \frac{J_1(\mu_m r/a)}{(ha^2)[1-(1/\mu_m)^2]J_1^2(\mu_m)} \frac{\mu_m}{2a} S_m(z) \tag{22}$$

$$S_m = \begin{cases} \frac{cosh(\mu_m z/a)cosh(\mu_m(h-z_0)/a)}{(\mu_m/a)sinh(\mu_m h/a)}, & z < z_0 \\ \frac{cosh(\mu_m(h-z)/a)cosh(\mu_m z_0/a)}{(\mu_m/a)sinh(\mu_m h/a)}, & z \geq z_0 \end{cases} \tag{23}$$

where h is the size of the cylinder in z, a is the size in r, and μ_m is the mth root of $J_1'(x) = 0$.

The second approach is to expand as a Fourier series in z, then solve the resulting differential equation in r for each component. This results in the equation:

$$\psi_2(r,z) = S\sum_{k=0}^{\infty} cos(\frac{\pi k}{h}z)cos(\frac{\pi k}{h}z_0)\frac{\pi k}{h^2}\left(K_1(\frac{\pi k}{h}r) - I_1(\frac{\pi k}{h}r)\frac{K_1'(\frac{\pi k}{h}a)}{I_1'(\frac{\pi k}{h}a)}\right) + S\frac{r/a^2 + 1/r}{2h} \tag{24}$$

Each of these methods has a part of the domain in which convergence is slow with respect to the number of terms in the series. The first method works poorly for $z \approx z_0$. The second method works poorly for $r \approx 0$. These solutions can be spliced together by applying the condition:

$$\text{If} \begin{cases} |z - z_0| \geq r, & \psi(r,z) = \psi_1(r,z) \\ |z - z_0| < r, & \psi(r,z) = \psi_2(r,z) \end{cases} \tag{25}$$

This results in a solution that converges rapidly at all points except the source, which is divergent to begin with. This yields a solution that can be used to benchmark a numerical model.

References

[1] M.V. Umansky, X.Q. Xu, B. Dudson, L.L. LoDestro, J.R. Myra, Computer Phys. Comm. **180**, 887 (2009).

[2] A. Zeiler, J. F. Drake and D. Biskamp, Phys. Plasmas **4**, 991 (1997).

[3] C. S. Chang et al., Journal of Physics: Conference Series **180** (2009) 012057.

[4] R. H. Cohen, and X. Q. Xu, Contrib. Plasma Phys. **48**, 212 (2008).

[5] D. R. Hatch, P. W. Terry, F. Jenko, F. Merz, and W. M. Nevins, Phys. Rev. Lett. **106**, 115003 (2011).

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

[7] W.L. Oberkampf and T. G. Trucano, Progress in Aerospace Sciences **38**, 209 (2002).

[8] D. A. Baver, J. R. Myra, M. V. Umansky, Comp. Phys. Comm. **182**, 1610 (2011).

[9] D.D. Ryutov, Phys. Plasmas **14**, 064502 (2007).

[10] H. Strauss, Phys. Fluids **24**, 2004 (1981).

[11] T. C. Hender, B. A. Carreras, W. A. Cooper, J. A. Holmes, P. H. Diamond, and P. L. Similon, Phys. Fluids **27**, 1439 (1984).

[12] P. N. Guzdar and J. F. Drake, Phys. Fluids B **5**, 3712 (1993).

[13] E. A. Belli, J. Candy, Phys. Plasmas **17**, 112314 (2010).

[14] http://www.grycap.upv.es/slepc/

[15] http://www.mcs.anl.gov/petsc/

[16] J. R. Shewchuk, Delaunay Refinement Algorithms for Triangular Mesh Generation, Computational Geometry: Theory and Applications **22**(1-3):21-74, May 2002.

[17] J. W. Connor and J. B. Taylor, Phys. Fluids **30**, 3180 (1987)

Figure captions

Table 1: Variant builds of the ArbiTER code. A and B represent eigenvalue matrices, λ represents eigenvalues, x represents eigenvectors, np is number of processors.

Figure 1: Relationship between components of the ArbiTER code.

Figure 2: Block structure of 2DX emulation topology for ArbiTER.

Figure 3: Results from simulation of resisstive ballooning modes in snowflake geometry. The upper plots show the distribution of mode amplitude. The lower plots show the corresponding density profiles used in the above simulations.

Figure 4: Results from simulation of a kinetic ballooning mode. Results are compared to fluid ballooning models and also to an iterative semi-fluid method used with the 2DX code.

Figure 5: Fourth eigenmode from a finite-element simulation of a diffusion/wave equation with zero-derivative boundary conditions. Color and height correspond to function value. Mesh used is superimposed onto solution.

Figure 6: Comparison of analytic and numerical solutions of a point magnetic dipole in a superconducting cylinder.

Figure 7: Layout of regions used in the extended domain method.

Figure 8: Eigenfunction from a resistive ballooning model, with extended domain solution superimposed onto 2D space.

Figure 9: Eigenfunction from a resistive ballooning model in the extended domain method, in the original extended domain grid.

Variant	Outputs	Uses
arbiter	λ and x	Solving eigenvalue problems
arbiterp	λ and x in np files	Solving eigenvalue problems in parallel
arbiterm	A and B	Debugging, interfacing with stand-alone eigensolver
arbiterog	all data except above	Debugging
arbitersd	x from $Ax = Bb$	Source-driven problems
arbiterdi	x from $x = Ab$	Measuring effect of specific terms in equation
arbiterfd	functions, constants, domains	Iterative generation of grid files

Table 1:

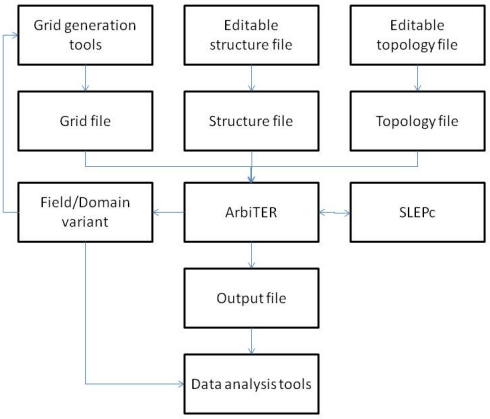

Figure 1:

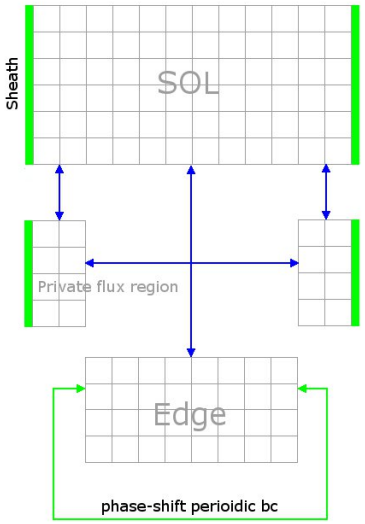

Figure 2:

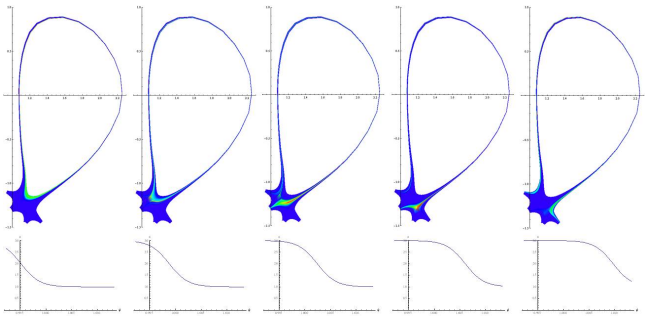

Figure 3:

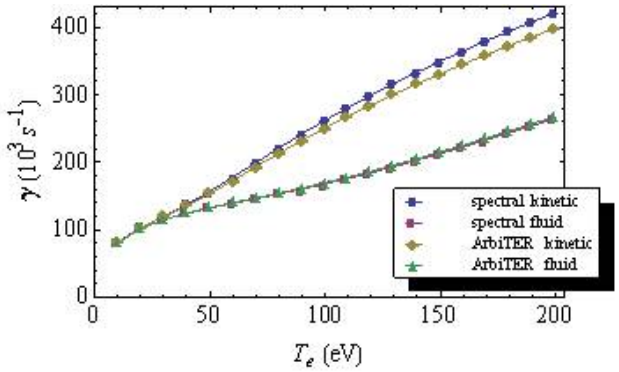

Figure 4:

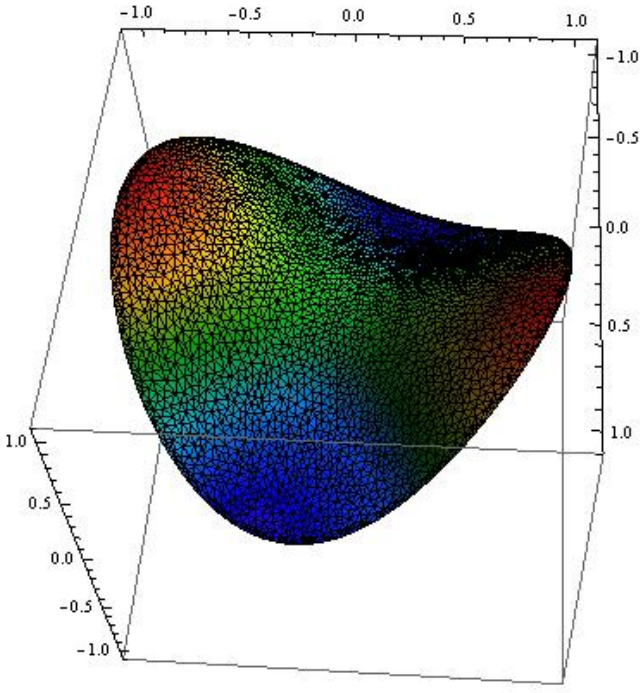

Figure 5:

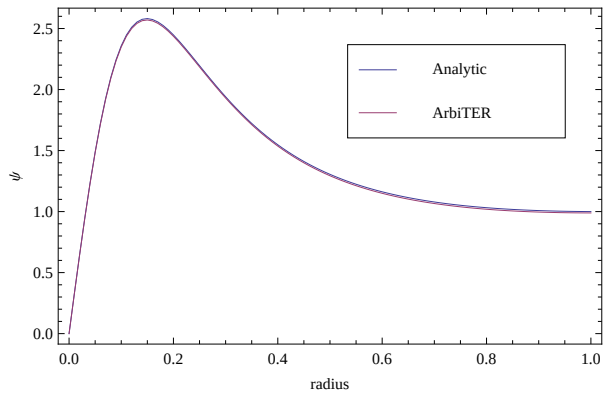

Figure 6:

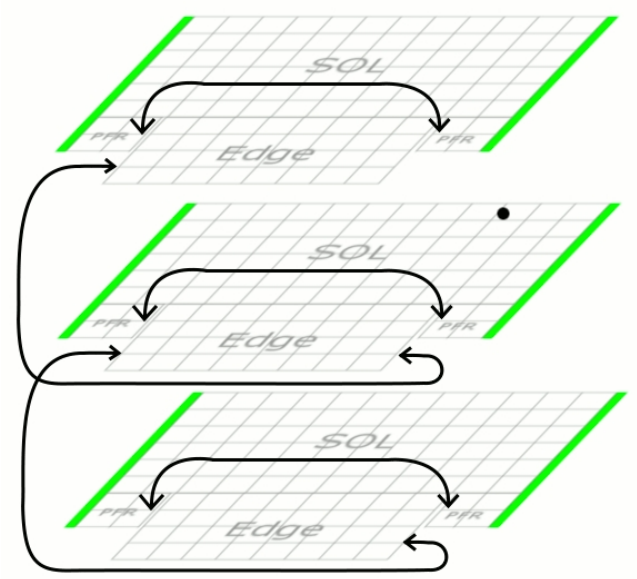

Figure 7:

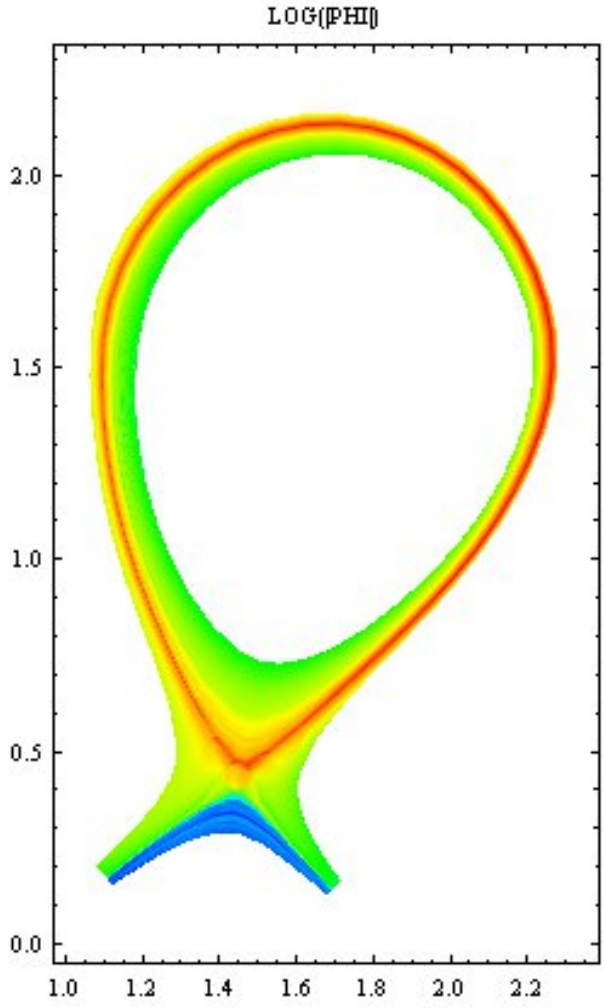

Figure 8:

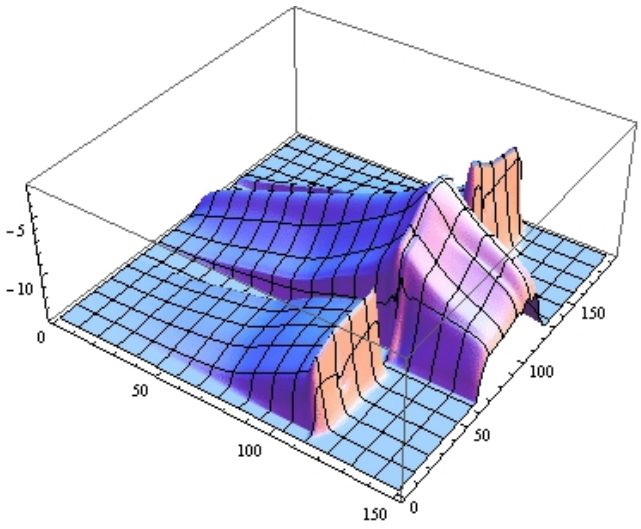

Figure 9: