UK Atomic Energy Authority

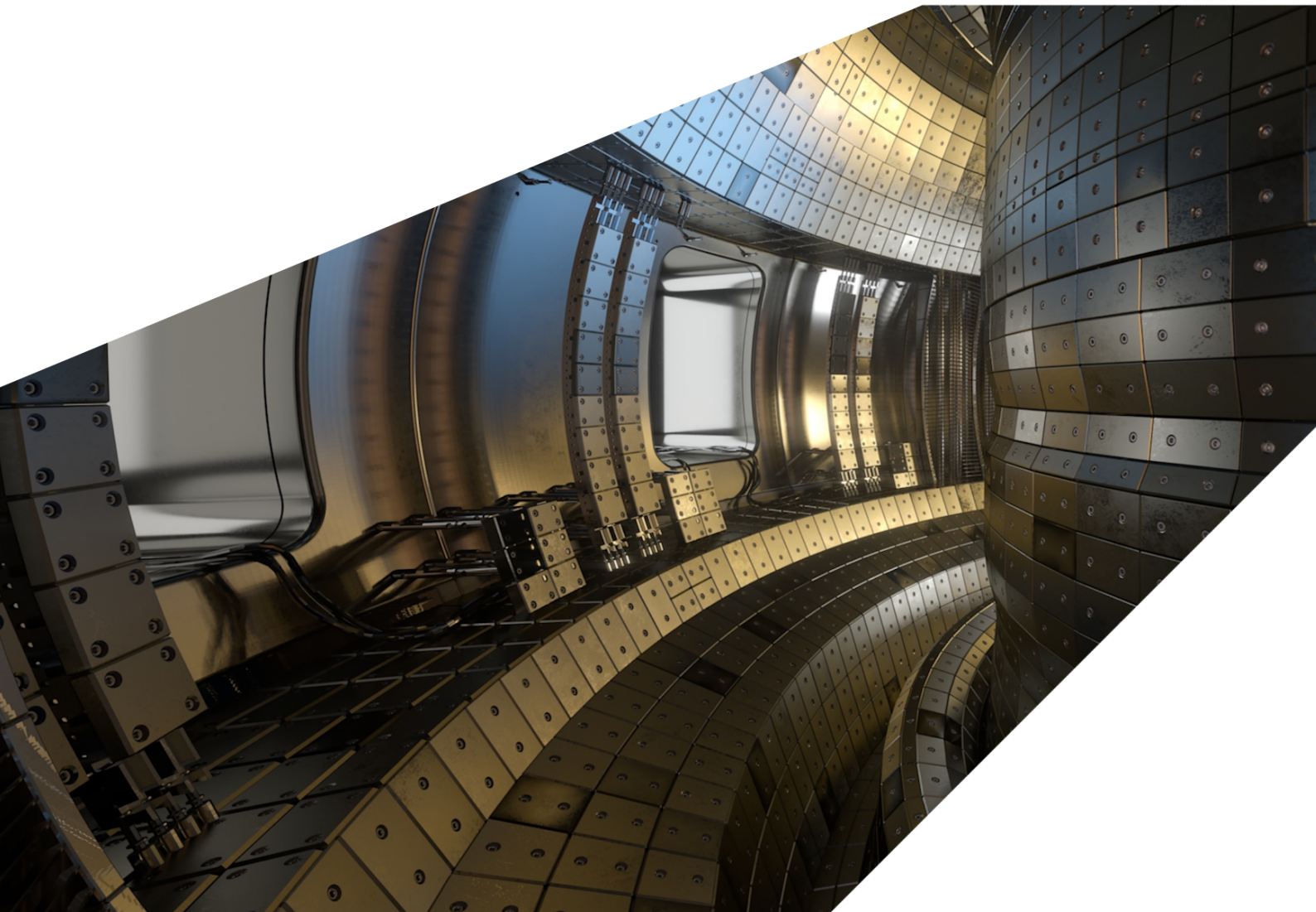
ExCALIBUR

Complementary actions. Code integration, acceptance and operation 4.

M6c.5 Version 1.00

Abstract

The report describes work for ExCALIBUR project NEPTUNE at Milestone M6c.5. It provides material related to the implementation of 3-D plasma turbulence equations using finite element modelling (FEM) techniques. Results produced with the Firedrake FEM sofware are presented, demonstrating its usefulness as a tool for prototyping new solver implementations and verifying the correctness of existing NEPTUNE proxyapps. An implementation of the 3-D Hasegawa-Wakatani equations in the Neptune Exploratory SOfware (NESO) is introduced and preliminary results are reported. A new code for generating field-aligned computational meshes is discussed and example output is presented in 2-D and 3-D for realistic tokamak geometries. Finally, a synopsis of related work carried out by grantees over the past year is provided in the form of technical report summaries and reviews of code deliverables.



UKAEA REFERENCE AND APPROVAL SHEET

	Client Reference:	
	UKAEA Reference:	CD/ExCALIBUR -FMS/0084
	Issue:	1.01
	Date:	13 March 2024
Project Name: ExCALIBUR Fusion Modelling System		

	Name and Department	Signature	Date
Prepared By:	Ed Threlfall Owen Parry Chris MacMackin CD	N/A N/A N/A	13 March 2024 13 March 2024 13 March 2024
Reviewed By:	James Cook CD		15 March 2024

1 Introduction

This report covers a range of topics related to the use of finite element modelling frameworks in NEPTUNE. Section 2 focuses on the use of the Firedrake[1] software as a tool for rapid prototyping of new solvers and as a means of checking the correctness of existing proxyapps. In addition to reimplementing some of those proxyapps and comparing to the Nektar++ versions directly, Firedrake is also employed to investigate various key components of a more complex model, which represents the near-term goal of fluid plasma modelling in the NEPTUNE Exploratory Software (NESO).

Section 3 reports on an implementation of the 3-D Hasegawa-Wakatani equations in NESO. The solver extends an existing Nektar++ equation system, but has required substantial code refactoring in order to adhere to sound software design practices, avoiding code duplication, preserving encapsulation and allowing key functionality to be shared between similar solvers.

An important component of project NEPTUNE is the development of technology to produce computational meshes that are suitable for modelling plasma and neutrals in the tokamak edge region. A related challenge is the high degree of velocity anisotropy associated with magnetically confined plasmas, which can negatively impact numerical accuracy when a mesh is not aligned with the magnetic field. To this end, Section 4 describes a tool called "NESO-fame" which has been developed to produce field-aligned meshes in realistic geometries (i.e. conforming to the first wall of a tokamak). A number of example meshes are presented in 2-D and 3-D.

Finally, Section 5 summarises related technical reports and code deliverables produced by project grantees in the past year. A series of progress reports are provided by the Oxford grantee, describing important updates to their implementation of a finite element drift-kinetic model. The KCL grantee work includes a report outlining improvements to their mesh generation tool, NekMesh, and several code deliverables designed to increase the usability and performance portability of Nektar++, as well as increase its utility for modelling fusion-relevant problems.

2 Firedrake as a tool to assist NEPTUNE development

The aim of the work described in this section has been to replicate NEPTUNE proxyapps using Firedrake[1], for verification and validation purposes, to enable the comparison of different frameworks for NEPTUNE finite-element aspects, to assist with finding bugs in existing NEPTUNE proxyapps, and also to speed up development. It will be demonstrated that many of the finite-element-based proxyapps developed under NEPTUNE may straightforwardly be implemented in the Firedrake/Irksome[2] framework.

As explained in [1], Firedrake is a framework that automates the finite-element solution of partial differential equations. The software user specifies the equation in a high-level, mathematical domain-specific language called Unified Form Language (UFL), in which PDEs are expressed in weak form, and then the package generates C code to solve the system, using also the PETSc library. Time-stepping is provided by the complementary Irksome package which offers a range of time-steppers within the Firedrake framework (though it is also possible to implement time-steppers without Irksome, see e.g. [3]). One major attraction of Firedrake is the ease of im-

plementation, which gives a favourable time-to-working-solver for many problems; it is, however, the intention of the developers of Firedrake that it be a competitive HPC tool, for example it has been tested up to approximately 25,000 CPU cores (cf. Nektar++, which has been run on up to c.100,000 cores). Another attraction is the Firedrake Github Discussions forum, which the author has found a useful and responsive source of guidance.

There are other tangible benefits from Firedrake such as the easy interchangeability of numerical methods e.g. time-steppers and preconditioners, with obvious synergies with the work done under the numerical analysis procurement (currently T/AW087/22). It is well-known that these aspects can yield large increases in computational efficiency for a relatively small investment in development time (see e.g. S.3.1.3 of [4]). The investigation of numerical methods is to an extent independent of the finite-element framework used - one may as well select one's tool on the basis of expediency - and to an extent not - why spend time implementing / debugging a promising method in one framework if it is available already elsewhere.

Note that the problems treated here are small (reflecting the state of the current NEPTUNE proxyapp ecosystem), though they can be extended to truly HPC-grade cases e.g. by moving to three dimensions and higher numerical resolution. Even the small problems can benefit from HPC e.g. via running ensembles for UQ or surrogate construction, or by a parallel-in-time implementation (the latter has been implemented for the Nektar-Driftwave proxyapp [5]).

Note also that, at the time of writing, little quantitative testing has been done on the outputs of the codes in the sequel (again a bit like the case for many of the existing NEPTUNE proxyapps).

One extremely important example case is that of the Large PLasma Device (LAPD) simulation in Nektar++. Major components of the physics in that problem are a sonic outflow of a compressible fluid, and drift-wave turbulence in the transverse plane. The first few sections in the sequel address these cases. The implementation of the LAPD simulation in the NESO framework, using Nektar++ and described in [6], is currently still ongoing. The author of this section believes that the remaining issues with the implementation of the LAPD simulation will be resolved in the near future with the aid of Firedrake prototyping.

Note that the scripts used to generate the outputs in this section can be obtained by requesting access to the private repository [7].

2.1 One-dimensional outflow problem

The primary motivation for this section is to provide an initial implementation of the problem of sonic outflow of a compressible fluid, in order to facilitate the implementation of the LAPD simulation within Nektar++.

2.1.1 The model

A very simple 1D model for the scrape-off layer is to take the compressible isothermal gas dynamics of a single species with no electric field and no transverse dynamics. Note that the isothermal

approximation is used in some plasma simulations e.g. the Blob2D example at [8] and for the ions in the cold-ion approximation, see e.g. [9].

In the first instance for discussion, the equations are the steady-state of eqs.10 and 11 from [10]. Two ODEs on the domain $x \in [-1, 1]$, describe the fluid density n and the flow velocity u:

$$\begin{aligned} (nu)' &= \quad n^*, \\ (nu^2)' &= -Tn'. \end{aligned} \tag{1}$$

Prime is x-derivative. n^* is a spatially-constant density source (cf. the LAPD simulation in which there is assumed to be a longitudinally-constant source of electron density). The solution is, specifying sonic outflow at the boundaries,

$$\begin{aligned} n &= \frac{n^*}{\sqrt{T}} \left(1 + \sqrt{1 - x^2}\right), \\ u &= \sqrt{T} \left(\frac{1 - \sqrt{1 - x^2}}{x}\right). \end{aligned} \tag{2}$$

For the following numerics the choices $n^* = 1$, $T = 1$ will be made.

Note that an attempt was made to solve the same system including diffusion terms of various types but no analytic solution was found.

2.1.2 Numerical treatment: continuous Galerkin (CG)

The equation treated by Firedrake is input in weak form using the UFL interface. For the system Eqs.1, the UFL is shown in Listing 1.

Listing 1: UFL code defining the equations Eqs.1 for CG elements. Note that `v1` and `v2` are test functions.

```
a = (grad(n*u)[0]*v1-nstar*v1+grad(n*u*u)[0]*v2+T*grad(n)[0]*v2)*dx
```

A naive treatment of the above hyperbolic system is prone to grid-scale oscillations which arise from a negative numerical diffusivity coefficient (see, for example, Remark 2.6 of [11]). Thus the simple Firedrake treatment in Listing 1 of the steady-state has the undesirable appearance shown in Fig.1.

There are two main ways to fix the problem of grid-scale numerical oscillation, both of which are basically equivalent to adding in a positive numerical diffusivity. These respectively will be the subject of the next subsections.

2.1.3 Numerical treatment: streamline-upwind (SU) correction

The first method is to add numerical diffusivity by modification of the basis functions. The modification leads to additional terms in the weak form to account for added numerical damping; see

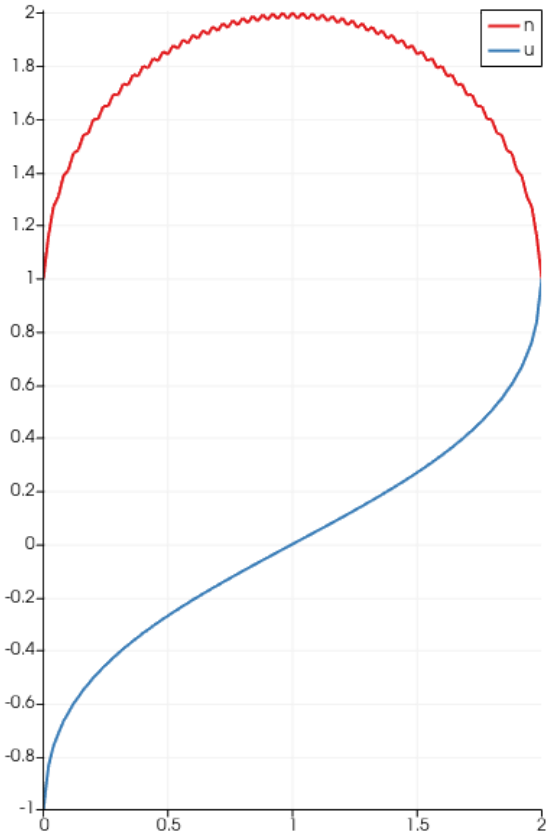


Figure 1: Numerical solution of Eqs.1 using 100 first-order CG elements. Note obvious grid-scale oscillation in n. Generated by `SOL_1D_CG_SU_DG_upwind.py` [7]; CG output.

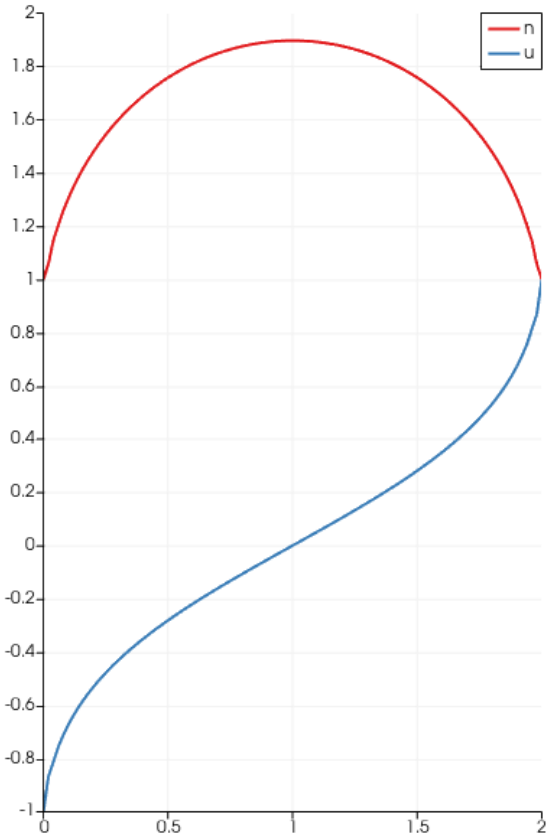


Figure 2: Numerical solution of Eqs.1 using 100 first-order CG elements with streamline-upwind (SU) correction. Note suppression of grid-scale oscillation. Generated by `SOL_1D_CG_SU_DG_upwind.py` [7]; CG output with SU correction enabled.

e.g. S.2.3 of [11]. The method demonstrated here will be referred to as a streamline-upwind (SU) correction.

Listing 2: UFL code defining the SOL1D equations for CG elements including the SU correction terms. Here h is the grid scale.

```
a = (grad(n*u)[0]*v1-nstar*v1+grad(n*u*u)[0]*v2+T*grad(n)[0]*v2)*dx \
  + (0.5*h*(grad(n)[0]-nstar)*grad(v1)[0])*dx \
  + (0.5*h*(grad(n*u)[0])*grad(v2)[0])*dx
```

This method correctly suppresses the grid-scale oscillation, as is seen in Fig.2.

It is obvious that this method has the weakness that it is accurate to the model only to first-order in the grid-spacing scale h - this is apparent from the $\mathcal{O}(h)$ correction to the weak form.

A time-dependent implementation is also possible; in this case the relevant equations are

$$\begin{aligned} \dot{n} + (nu)' &= n^*, \\ (\dot{nu}) + (nu^2)' &= -Tn'. \end{aligned} \tag{3}$$

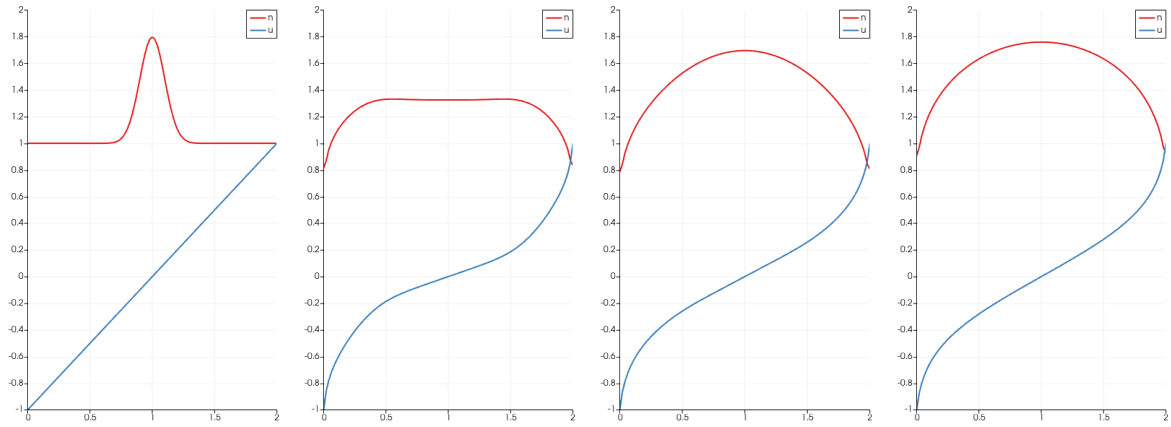


Figure 3: Numerical solution of eqs.3 using 100 first-order CG elements with SU correction starting from a Gaussian n and linear u initial condition. Plots from the left are $t = 0, 1, 2, 3$. Generated by `SOL_1D_SU_irksome.py [7]`.

Note that the second of these is recast in the form $n\dot{u} + un^* + nuu' = -Tn'$ in order to derive the weak form in Listing 3.

Listing 3: UFL code (including Irksome time-derivatives) defining the equation system Eqs.3 plus SU corrections.

```
F = ((Dt(n)*v1)*dx + (n*Dt(u)*v2)*dx) \
    + (grad(n*u)[0]*v1-v1*nstar)*dx \
    + (nstar*u*v2+n*u*grad(u)[0]*v2+Temp*grad(n)[0]*v2)*dx \
    + (0.5*h*(grad(n)[0]-nstar)*grad(v1)[0])*dx \
    + (0.5*h*(grad(n*u)[0])*grad(v2)[0])*dx
```

Output from the time-dependent implementation with SU correction weak form is exhibited in Fig.3. It does show what looks like a minor discrepancy in n near the boundaries.

2.1.4 Numerical treatment: discontinuous Galerkin (DG)

The second method of stabilizing the flow equations is to use a discontinuous Galerkin method with a dissipative numerical flux. Here an upwind flux is used (other choices, e.g. an interior penalty flux, are possible).

Listing 4: UFL code defining the equations Eqs.1 for DG elements with upwind numerical flux. Note that the backslashes denote new lines in the UFL whereas carriage returns without backslash are formatting to keep the listing text on the page.

```
aD = (nD*dot(uD, grad(v1D))+v1D*nstar)*dx \
    + (nD*uD[0]*dot(uD, grad(v2D[0]))+T*nD*grad(v2D[0])[0])*dx \
    - conditional(dot(uD, norm) > 0, v1D*dot(uD, norm)*nD, 0.0)*ds \
```

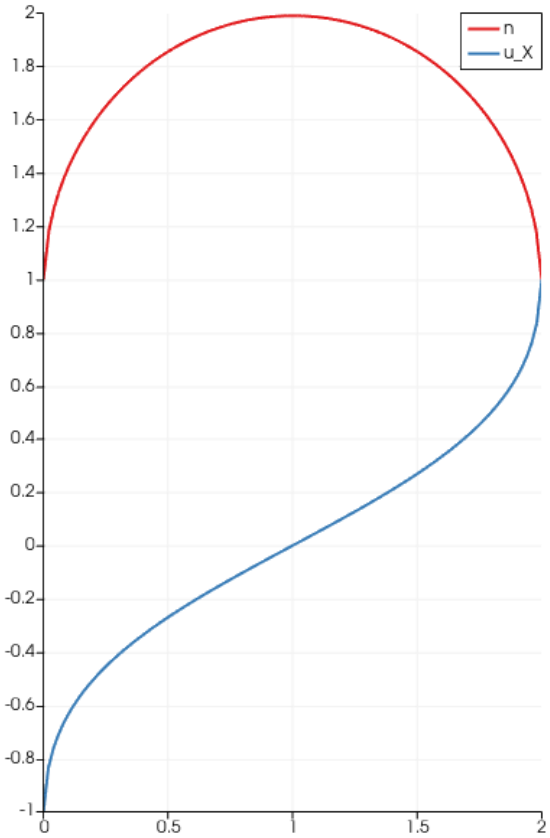


Figure 4: Numerical solution of eqs.1 using 100 first-order DG elements with upwind numerical flux. Note suppression of grid-scale oscillation. Generated by `SOL_1D_DG_upwind.py` [7].

```
  - (v1D('+') - v1D('-'))*(uD_n('+')*nD('+') - uD_n('-')*nD('-'))*dS \
  - (v2D('+')[0] - v2D('-')[0])*(nD('+')*uD('+')[0]*uD[0]('+') - nD('-')
        *uD('-')[0]*uD[0]('-'))*dS \
  - (v2D('+')[0] - v2D('-')[0])*(nD('+') - nD('-'))*T*dS
```

A disadvantage of the DG method is that the degree-of-freedom count is increased due to the duplication of boundary degrees of freedom. This problem is most acute for low orders, where boundary degrees of freedom dominate the computation. It is noted that the DG implementations are slower than the SU-corrected CG ones for the small codes described in this section of the report.

The DG method has the advantage that the mass matrix is defined on a per-element basis; the coupling between elements is exclusively via the element boundary numerical flux.

Output from the stationary-state and time-dependent DG implementations of the weak form in Listing 5 is shown in Figs.4, 5. These outputs look better-behaved than the SU version near the boundaries.

Listing 5: UFL code (including Irksome time-derivatives) defining the equation system Eqs.3 for upwind DG.

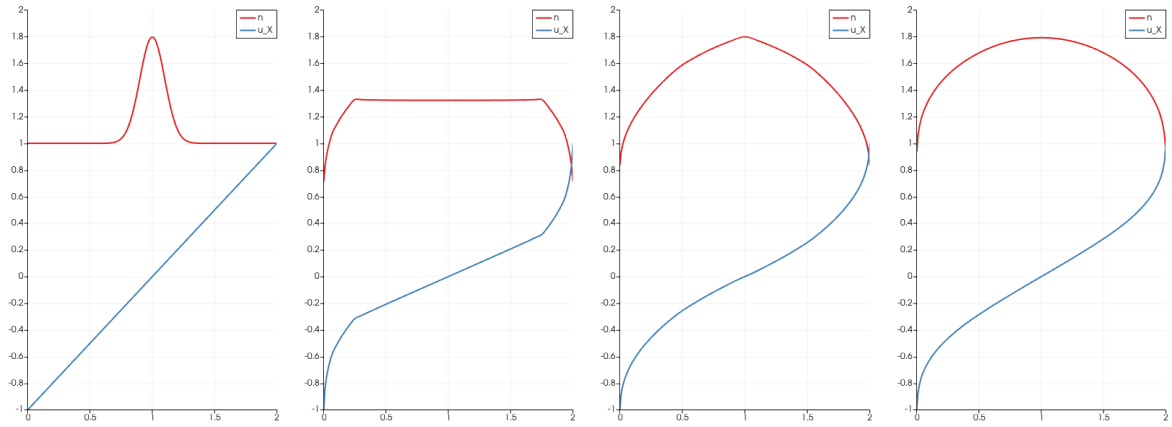


Figure 5: Numerical solution of eqs.3 using 100 first-order DG elements with an upwind numerical flux, starting from a Gaussian n and linear u initial condition. Plots from the left are $t = 0, 1, 2, 3$. Generated by `SOL_1D_DG_upwind_irksome.py [7]`.

```
F = -((Dt(n)*v1)*dx + (n*dot(Dt(u),v2))*dx) \
  + (n*dot(u, grad(v1))+v1*nstar)*dx \
  + (nstar*dot(u,v2)+n*u[0]*grad(dot(u,v2))[0]+n*u[0]*dot(u, grad(v2[0]))
        +Temp*n*grad(v2[0])[0])*dx \
  - (v1('+') - v1('-'))*(u_n('+')*n('+') - u_n('-')*n('-'))*dS \
  + (u('+')[0]*v2('+')[0]-u('-')[0]*v2('-')[0])
        *(n('+')*u_n('+')-n('-')*u_n('-'))*dS \
  - conditional(dot(u, norm) > 0, v1*dot(u, norm)*n, 0.0)*ds
```

Note also that scripts have been written to solve the above problem on 2D and 3D domains in anticipation of elaborating this simple case into a script including the physics in the 3D LAPD simulation; the density outputs are shown in Fig.6.

2.2 Drift-wave turbulence in Firedrake

The primary motivation for this section is to provide an initial implementation of drift-wave turbulence in the transverse plane that can be integrated with the longitudinal outflow dynamics explained above, in order to facilitate the implementation of the LAPD simulation within Nektar++. Firedrake is again used here as an expedient framework in which to implement test cases. A secondary motivation for this work is to supplement the existing NEPTUNE 2D turbulence proxyapp, the Nektar++-based Nektar-Driftwave [5], with an implementation in Firedrake / Irksome in which the details of the numerical method, e.g. the choice of finite-element discretization and the choice of time-stepper can be easily changed. (*Apologia* - extensive investigation of the effect of varying the numerical method has not at the time of writing been performed but it could straightforwardly be done, along with the addition of quantitative diagnostics such as energy, enstrophy, and power spectra of the turbulent flow. Also, non-conservation of ω leads to obvious problems solving the Poisson equation in the periodic case and so explicit conservation of ω is desirable. The author

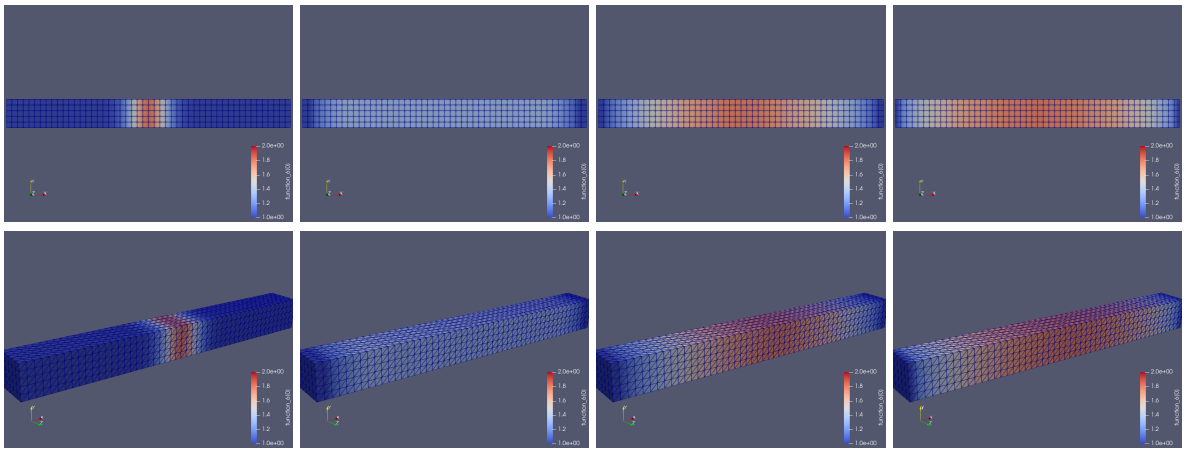


Figure 6: Numerical solution of eqs.3 in 2D (top) and 3D (bottom), using first-order DG elements with starting from a Gaussian n and linear u initial condition. Plots from left are $t = 0, 1, 2, 3$ in both cases. Generated by `SOL_2D_DG_upwind_irksome.py` [7] and `SOL_3D_DG_upwind_irksome.py` [7].

thereby justifies the rather heuristic and visually-led nature of the following presentation. Note also that performance measurements (wall times) for the codes do not reflect the scaling expected for larger problems and are quoted in the context of obtaining a workable code that runs rapidly in order to speed-up testing and progress toward more realistic problem e.g. the LAPD simulation.)

Nektar-Driftwave implements the Hasegawa-Wakatani equations, describing drift-wave turbulence, for density n, vorticity ω, and potential ϕ:

$$\begin{aligned} \frac{\partial n}{\partial t} + [\phi, n] &= \alpha(\phi - n) \\ \frac{\partial \omega}{\partial t} + [\phi, \omega] &= \alpha(\phi - n) - \kappa \frac{\partial \phi}{\partial y} \\ \nabla_\perp^2 \phi &= \omega. \end{aligned} \tag{4}$$

In the above, the 'Poisson bracket' denotes $[a, b] \equiv \frac{\partial a}{\partial x}\frac{\partial b}{\partial y} - \frac{\partial b}{\partial x}\frac{\partial a}{\partial y}$.

The model is identical to that used in the existing Nektar-Driftwave proxyapp. The example supplied with Nektar-Driftwave propagates the equations, starting from a Gaussian-profile density perturbation, and a vorticity perturbation. The vorticity integrates to zero (or very nearly zero depending on the condition that the vorticity amplitude drops to very nearly zero at the domain boundaries), which is a property that must be preserved by periodic simulations as the vorticity plays the role of a charge density. The simulations lasts for 50 time units using a timestep of 0.0005 time units, which is close to the stability limit of the explicit time evolution method used (standard RK4). The small timestep means that the simulation is quite slow, taking eight hours to run on an eight-core workstation (note that this uses a 64×64 mesh of square $p = 3$ DG elements).

Note that we fix the parameters $\kappa = 1$ (strength of driving force from background gradient) and $\alpha = 1$ (called adiabaticity and related to the coupling to the out-of-plane dynamics in the 3D form of the equations). The Firedrake examples in this section used $\kappa = 2$ and $\alpha = 2$ i.e. a larger driving force; this was done in order to ensure that the more strongly-damped cases evolved to a

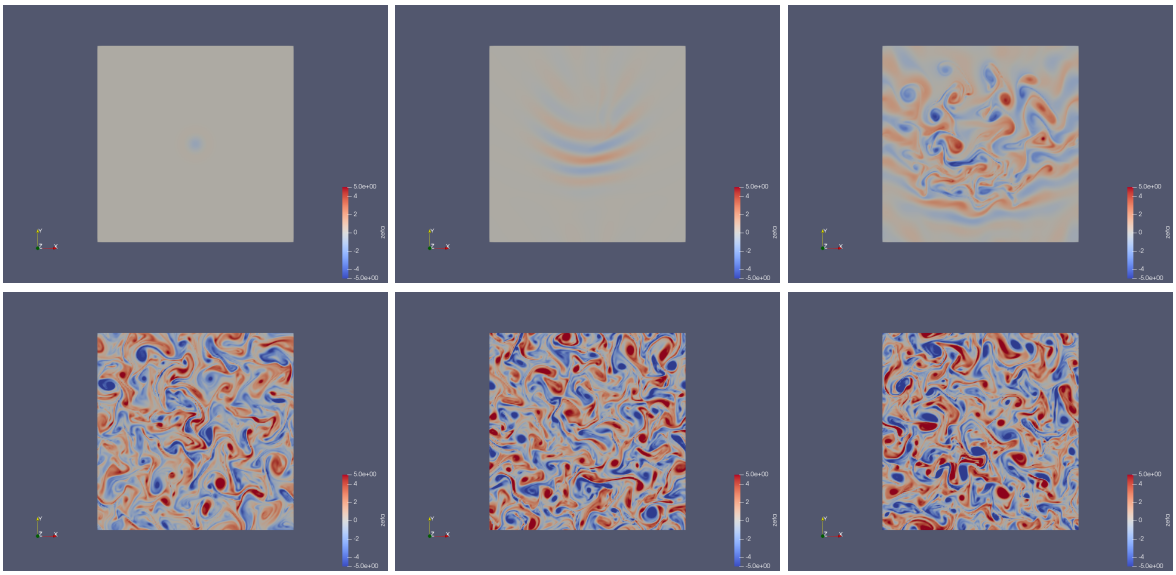


Figure 7: Numerical solution of the vorticity from Eqs.4 from Nektar-Driftwave using a 64×64 mesh of square third-order DG, starting from a Gaussian density perturbation initial condition. Plots from left-to-right then top-to-bottom are $t = 0, 10, 20, 30, 40, 50$. Note that these outputs are not directly comparable to the subsequent outputs from Firedrake due to the choice of different parameters; however the appearance of these is a useful guide to the lengthscales resolved in the turbulent state.

turbulent state. None of the examples in this section were found to need a hyperdiffusion term (in all cases there is sufficient numerical damping to keep the time-evolution stable). The initial data used in the 2D Firedrake examples was the same as that used in Nektar-Driftwave. The domain is 40×40 units in size and it has periodic boundary conditions on all boundaries.

Note that the equation system can be solved in one pass as a differential-algebraic equation (DAE). An alternative, which is the one to which we will devote the most attention, is to update the potential independently from the other variables in a leap-frogging fashion. This latter is the approach used in the existing Nektar-Driftwave proxyapp, though it is the case that the potential is solved at each stage of the RK stepper, using the vorticity field from the previous stage. Solving the potential on a per-stage basis in this way in Irksome would require additional work beyond that presented here.

A word on the current status of Nektar-Driftwave: the grantee (T/AW085/22) has recently made great progress in improving the efficiency of this solver by replacing the explicit RK4 time-stepping with an explicit method. There is also a parallel-in-time implementation, which the author understands was the movitation behind the implicit implementation. It is, however, not clear whether the performance gains will translate well to other plasma physics examples without careful consideration of the preconditioning of the system (the simple 2D Hasegawa-Wakatani implementation is found to work well without preconditioning).

2.2.1 Dirichlet boundary conditions

Outputs of a SU-corrected CG version (Listing 6) are shown in Fig.7.

Listing 6: UFL code defining the (non-elliptic part of the) equations for the 2D Hasegawa-Wakatani system for CG elements with SU correction.

```
F = Dt(w)*v1*dx + Dt(n)*v2*dx \
  - v1*div(w*driftvel)*dx - v2*div(n*driftvel)*dx \
  - alpha*(phi_s-n)*(v1+v2)*dx \
  + kappa*grad(phi_s)[1]*v2*dx \
  + 0.5*h*(dot(driftvel, grad(w))-alpha*(phi_s-n))*dot(driftvel, grad(v1))
        *(1/sqrt((driftvel[0])**2+(driftvel[1])**2))*dx \
  + 0.5*h*(dot(driftvel, grad(n))-alpha*(phi_s-n)-kappa*grad(phi_s)[1])*dot(drif
        *(1/sqrt((driftvel[0])**2+(driftvel[1])**2))*dx
```

Note that the size of the SU term has been reduced below the amount specified by the textbook [11], by a factor of ten. It was found that this led to a better resolution of the field activity without showing a large increase in grid-scale oscillation.

Using $p = 3$ elements for the vorticity and $p = 2$ for density, potential and drift velocity gives a simulation that takes approximately 21 minutes on one core of the author's laptop.

Outputs of a DG implementation (Listing 7) are shown in Fig.9. Note that only the advected fields are DG; CG was used for ϕ and drift velocity, as in Nektar-Driftwave and e.g. [12].

Listing 7: UFL code defining the (non-elliptic) equations for the 2D Hasegawa-Wakatani system using upwind DG elements.

```
norm = FacetNormal(mesh)
driftvel_n = 0.5*(dot(driftvel, norm)+abs(dot(driftvel, norm)))

F = Dt(w)*v1*dx + Dt(n)*v2*dx \
  - v1*div(w*driftvel)*dx - v2*div(n*driftvel)*dx \
  - alpha*(phi_s-n)*(v1+v2)*dx \
  + kappa*grad(phi_s)[1]*v2*dx \
  + driftvel_n('-')*(w('-') - w('+'))*v1('-')*dS \
  + driftvel_n('+')*(w('+') - w('-'))*v1('+')*dS \
  + driftvel_n('-')*(n('-') - n('+'))*v2('-')*dS \
  + driftvel_n('+')*(n('+') - n('-'))*v2('+')*dS
```

Using $p = 3$ elements for the vorticity and $p = 2$ for density, potential and drift velocity gives a simulation that takes approximately 100 minutes on one core of the author's laptop. This is significantly longer than the SU-corrected CG implementation, but the results to seem to indicate a higher resolution of the turbulent flow.

A DG-DAE implementation was created (Listing 8). This was run for 50 time units using 500 timesteps. The timestepper was RadauIIA(2). The output shows considerably more resolution

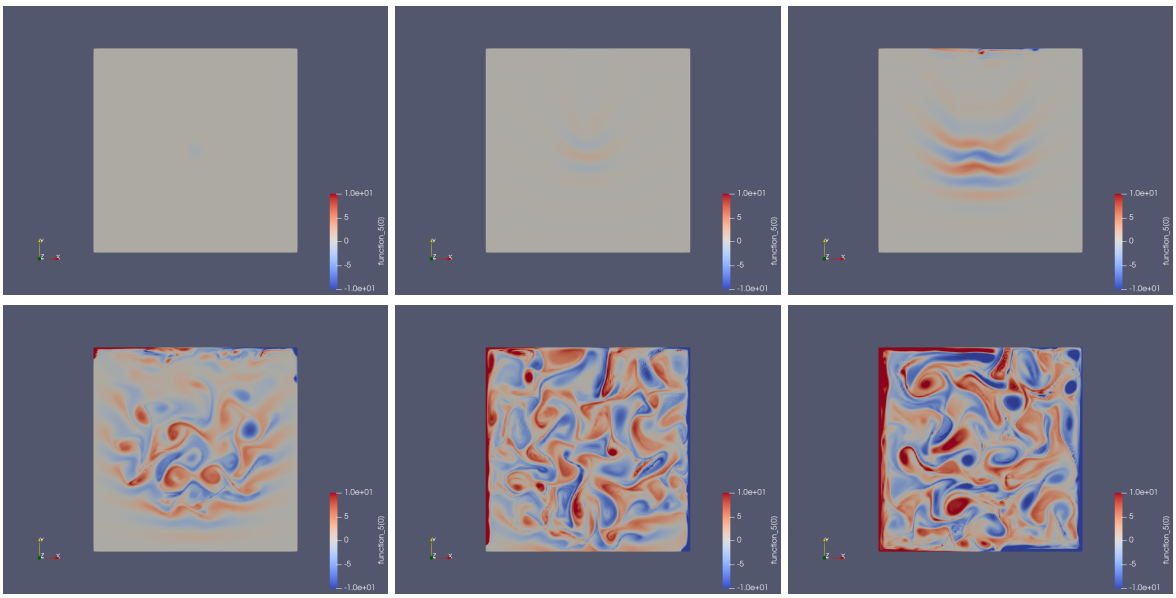


Figure 8: Numerical solution of the vorticity from Eqs.4 using a 64×64 mesh of square first-order CG elements with a SU correction term, starting from a Gaussian density perturbation initial condition. Plots from left are $t = 0, 10, 20, 30, 40, 50$. Note that in subsequent plots, the first three times are not shown as they are largely identical to those in the above (the system is in a laminar regime in which the numerical methods give visually very similar results). Note the boundary effects caused by the use of a homogeneous Dirichlet boundary condition for the potential (and homogeneous Neumann for the other variables). Generated by `Nektar-Driftwave_port_irksome_SU.py` [7].

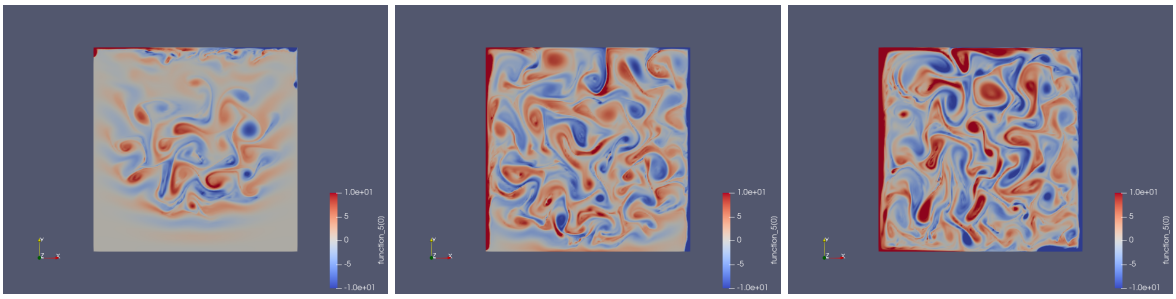


Figure 9: Numerical solution of the vorticity from Eqs.4 using a 64×64 mesh of square DG elements with upwind numerical flux, starting from a Gaussian density perturbation initial condition. Plots from left are $t = 30, 40, 50$. Generated by `Nektar-Driftwave_port_irksome_DG.py` [7].

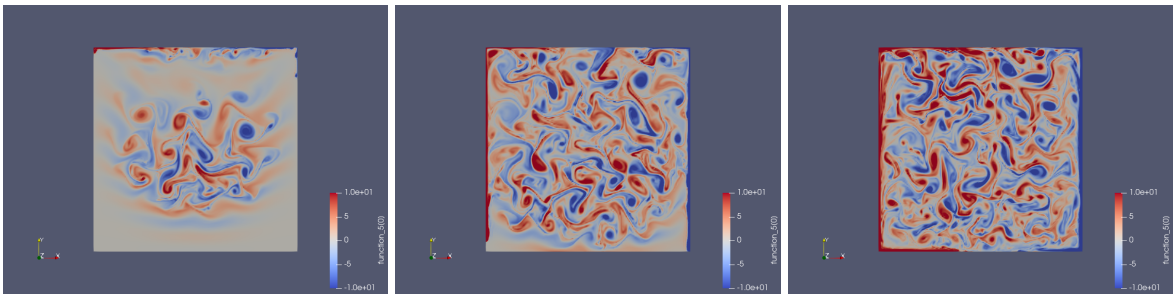


Figure 10: Numerical solution of the vorticity from Eqs.4 using a 64×64 mesh of square DG elements with upwind numerical flux and solving the elliptic equation for ϕ concurrently with the rest of the system, starting from a Gaussian density perturbation initial condition. Plots from the left are $t = 30, 40, 50$. Generated by `Nektar-Driftwave_port_irksome_DG_DAE.py` [7].

than the preceding examples, implying that the resolved scale depends on the timestep (and quite likely the accuracy implied by the scheme for updating ϕ). This implementation was found to be rather slow, taking approximately 24 hours to execute using a single core of a laptop. This implementation at least means that fully higher-order time-stepping can be accessed without further doctoring.

Listing 8: UFL code defining the equations for the 2D Hasegawa-Wakatani system including the elliptic solve (DG for advected fields and CG for potential solve).

```
norm = FacetNormal(mesh)

F = Dt(w)*v1*dx + Dt(n)*v2*dx \
  - v1*div(w*as_vector([grad(phi)[1], -grad(phi)[0]]))*dx
        - v2*div(n*as_vector([grad(phi)[1], -grad(phi)[0]]))*dx \
  - alpha*(phi-n)*(v1+v2)*dx \
  + kappa*grad(phi)[1]*v2*dx \
  + 0.5*(dot(as_vector([grad(phi)[1], -grad(phi)[0]]), norm)
        +abs(dot(as_vector([grad(phi)[1], -grad(phi)[0]]), norm)))('-')
        *(w('-') - w('+'))*v1('-')*dS \
  + 0.5*(dot(as_vector([grad(phi)[1], -grad(phi)[0]]), norm)
        +abs(dot(as_vector([grad(phi)[1], -grad(phi)[0]]), norm)))('+')
        *(w('+') - w('-'))*v1('+')*dS \
  + 0.5*(dot(as_vector([grad(phi)[1], -grad(phi)[0]]), norm)
        +abs(dot(as_vector([grad(phi)[1], -grad(phi)[0]]), norm)))('-')
        *(n('-') - n('+'))*v2('-')*dS \
  + 0.5*(dot(as_vector([grad(phi)[1], -grad(phi)[0]]), norm)
        +abs(dot(as_vector([grad(phi)[1], -grad(phi)[0]]), norm)))('+')
        *(n('+') - n('-'))*v2('+')*dS \
  + inner(grad(phi),grad(v3))*dx \
  + w*v3*dx
```

The first step towards extending this model into a three-dimensional plasma fluid simulator is to

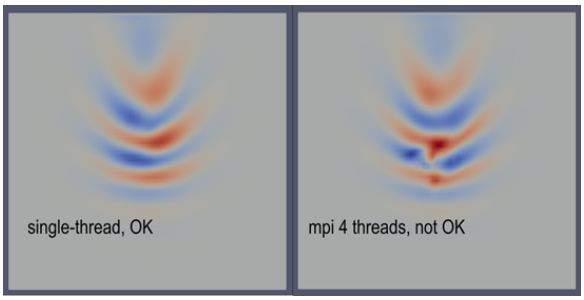


Figure 11: Illustration showing incorrect output from Firedrake when running with four MPI processes. Field on surface of 3D domain shown.

put the 2D dynamics in a 3D domain. Note that this may or may not result in a system symmetric in the new dimension, depending on the boundary conditions. A CG / SU version was implemented. In the initial version of this script, an inconsistency was found when running with MPI (Fig.11). This was resolved by dialogue with the developers; the problem is the discontinuous value of the facet-tangential derivative of the potential: the interpolation was giving a different answer in MPI depending on which cell was last evaluated. The solution is to avoid interpolating the drift velocity onto a Firedrake function but rather to substitute the expression directly into where it is needed. In addition, it was found that a small softening length needed to be added into the denominator of the SU stabilization terms, else the potential solver would crash immediately. Outputs from the working solver are shown in Figs.12 and 13.

Listing 9: UFL code defining the equation system for a MPI-capable implementation of the 2D Hasegawa-Wakatani equations in 3D. Note softening length in denominator.

```
F = Dt(w)*v1*dx + Dt(n)*v2*dx \
  - v1*div(w*as_vector([grad(phi_s)[1],-grad(phi_s)[0], 0.0]))*dx
        - v2*div(n*as_vector([grad(phi_s)[1],-grad(phi_s)[0], 0.0]))*dx \
  - alpha*(phi_s-n)*(v1+v2)*dx \
  + kappa*grad(phi_s)[1]*v2*dx \
  + 0.5*h*(dot(as_vector([grad(phi_s)[1],-grad(phi_s)[0], 0.0]), grad(w))
        - alpha*(phi_s-n))
        *dot(as_vector([grad(phi_s)[1],-grad(phi_s)[0], 0.0]), grad(v1))
        *(1/sqrt((grad(phi_s)[1])**2+(grad(phi_s)[0])**2+0.0001))*dx \
  + 0.5*h*(dot(as_vector([grad(phi_s)[1],-grad(phi_s)[0], 0.0]), grad(n))
        - alpha*(phi_s-n)-kappa*grad(phi_s)[1])
        *dot(as_vector([grad(phi_s)[1],-grad(phi_s)[0], 0.0]), grad(v2))
        *(1/sqrt((grad(phi_s)[1])**2+(grad(phi_s)[0])**2+0.0001))*dx \
```

2.2.2 Periodic boundary conditions

Two- and three- dimensional implementations with periodic boundary conditions have been written. The 2D case took approx. 58 minutes to run - slower than the equivalent Dirichlet-boundary simulation. Outputs are shown in Figs.14 and 15.

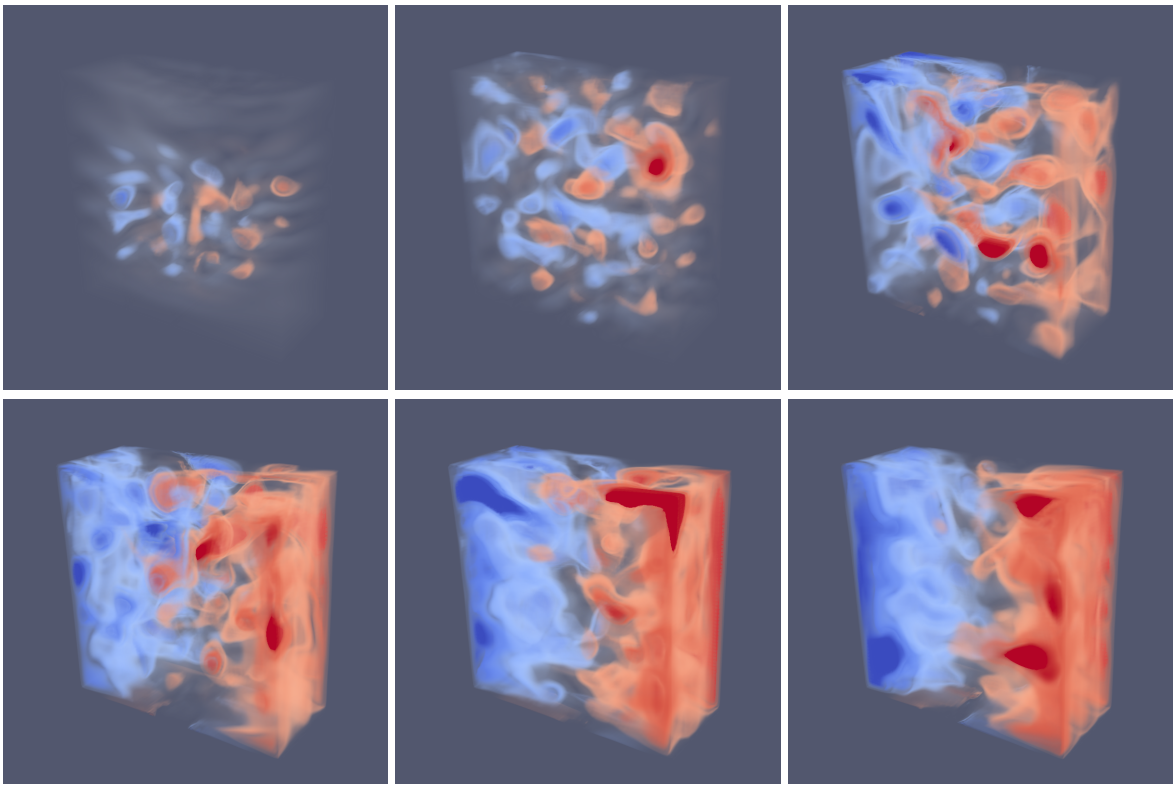


Figure 12: Numerical solution of Eqs.4, showing density, using a $64 \times 64 \times 16$ mesh further subdivided into tetrahedral first-order CG elements with a SU correction term, starting from a Gaussian density perturbation initial condition modulated by a sinusoid in the longitudinal direction. Density plotted using the Nvidia IndeX plugin in ParaView, using a custom non-monotonic colour map to emphasize the shape of the flow features. Plots from left-to-right then top-to-bottom correspond to $t = 0, 10, 20, 30, 40, 50$. Generated by `Nektar-Driftwave_port_irksome_SU_2Din3D.py` [7].

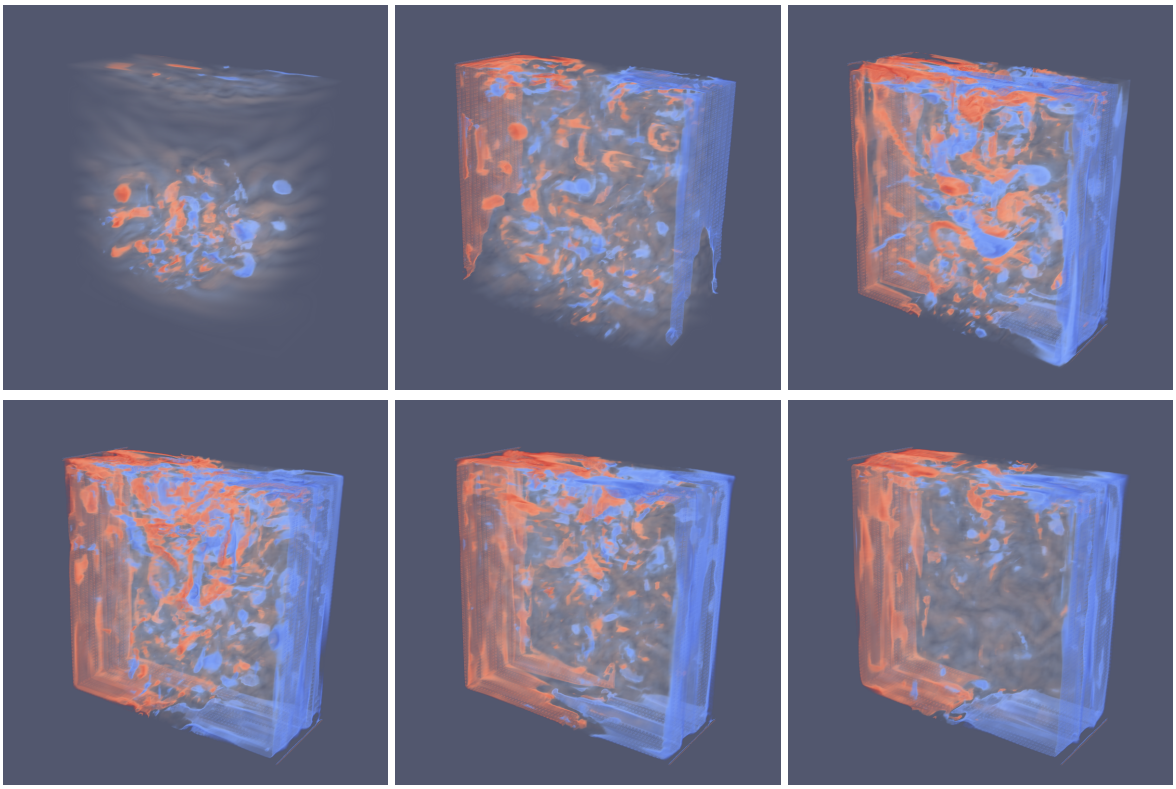


Figure 13: Numerical solution of Eqs.4, showing vorticity, using the same mesh as in Fig.12 of first-order CG elements with a SU correction term, starting from a Gaussian density perturbation initial condition modulated by a sinusoid in the longitudinal direction. Plots from left are $t = 0, 10, 20, 30, 40, 50$. Generated by `Nektar-Driftwave_port_irksome_SU_2Din3D.py` [7].

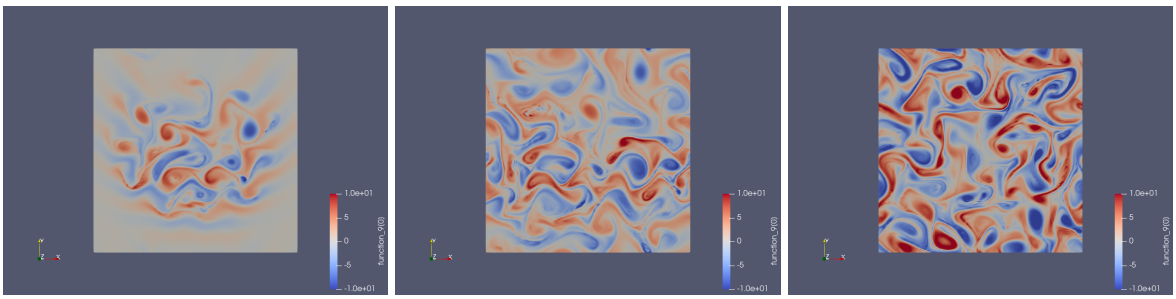


Figure 14: Numerical solution of Eqs.4, showing vorticity, using a periodic 64×64 mesh of square first-order CG elements with a SU correction term, starting from a Gaussian density perturbation initial condition. Plots from left are $t = 30, 40, 50$. Generated by `Nektar-Driftwave_port_irksome_SU_periodic.py` [7].

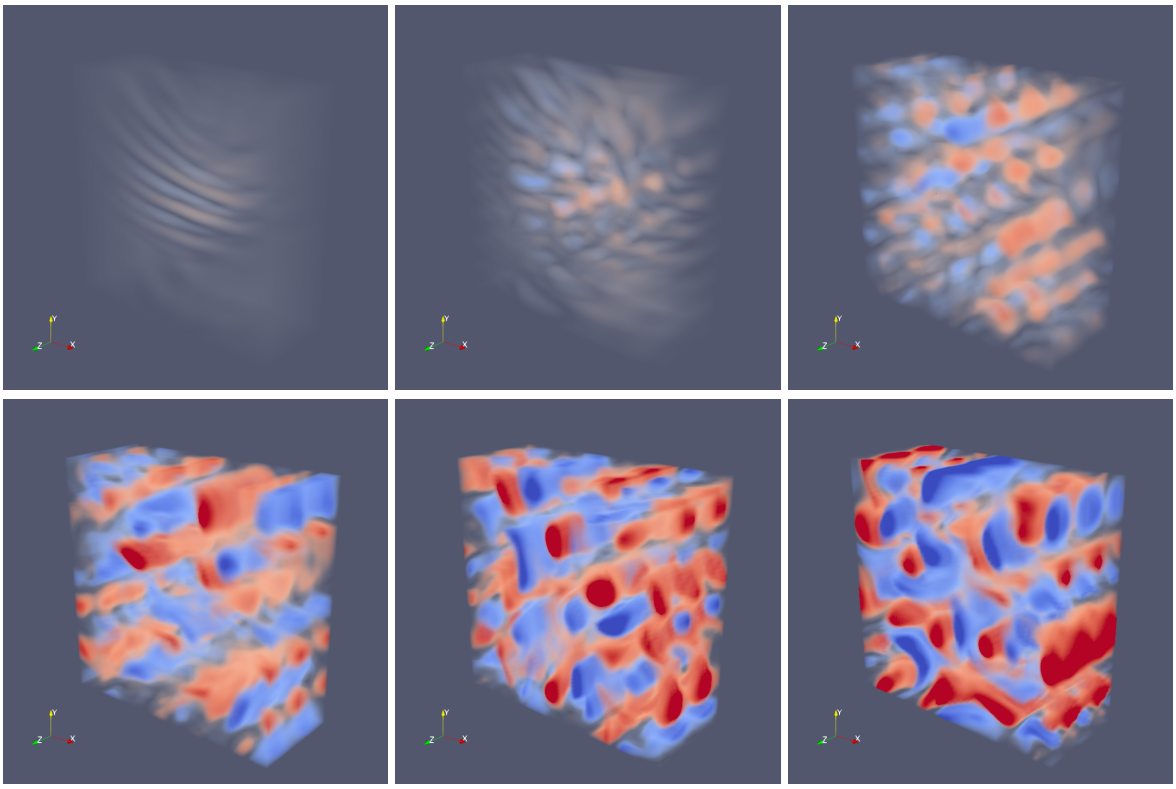


Figure 15: Numerical solution of Eqs.4, showing density, using a $64 \times 64 \times 16$ mesh of first-order CG elements with a SU correction term, starting from a Gaussian density perturbation initial condition modulated by a sinusoid in the longitudinal direction. Plots from left are $t = 0, 10, 20, 30, 40, 50$. Note symmetry. Generated by `Nektar-Driftwave_port_irksome_SU_2Din3D_periodic.py` [7].

2.2.3 Truly 3D Hasegawa-Wakatani

A version of the fully 3D Haswgawa-Wakatani system has been implemented and preliminary results have been obtained from it. (See `HW3D_SU_periodic.py`)

2.3 Blob2D example in Firedrake

The motivation for this work is to enable debugging of some of the examples implemented in Nektar++ that do not yet work correctly. There are two solvers representing the examples Blob2D-Te-Ti and 2D-drift-plane-turbulence-Te-Ti at [13]. In the present work, only the simplest example, Blob2D, is exhibited.

The explicit solver was constructed on the basis of mirroring closely the numerical implementation in the Nektar-Driftplane proxyapp. A higher-order explicit time-stepping scheme was chosen, after the implementation in [3] (note that this time-stepper does not use the Irksome package). A mesh of 64*32 square fourth-order DG elements was used, along with a timestep of $dt = 2 \times 10^{-3}$.

Listing 10: UFL code defining the equation system for the Hermes-3 Blob2D example.

```
L1 = -dt*( - v1*div(w*driftvel)*dx - v2*div(n*driftvel)*dx \
  - (phi_s*n/L_par)*(v1+v2)*dx \
  + kappa*grad(n)[1]*v1*dx \
  + driftvel_n('-')*(w('-') - w('+'))*v1('-')*dS \
  + driftvel_n('+')*(w('+') - w('-'))*v1('+')*dS \
  + driftvel_n('-')*(n('-') - n('+'))*v2('-')*dS \
  + driftvel_n('+')*(n('+') - n('-'))*v2('+')*dS )
```

The outputs (Fig.16) can be compared with those generated by the Nektar-Driftplane proxyapp; see Fig.2 of [14]. As in that case, the equations simulated here have no explicit diffusive terms (hence the behaviour is rather similar to that of an Euler fluid with the scale of feature generation being cut off by numerical diffusion). One advantage of Firedrake is that diffusion terms are easily added into the equations.

2.4 Plasma kinetics in Firedrake

2.4.1 Two-stream instability

The Vlasov-Poisson system is implemented in Firedrake. The code simulates the same two-stream instability as described in the report [15] in which a prototype Nektar++ solver was implemented. As noted in that report, the problem can be cast as an advection coupled to an elliptic solve, and it can be linearized by solving the elliptic problem and using that output to compute the advection velocity for a separate advection time step. The set-up involves two counter-propagating beams of like-charged particles on a periodic domain, along with a static neutralizing background

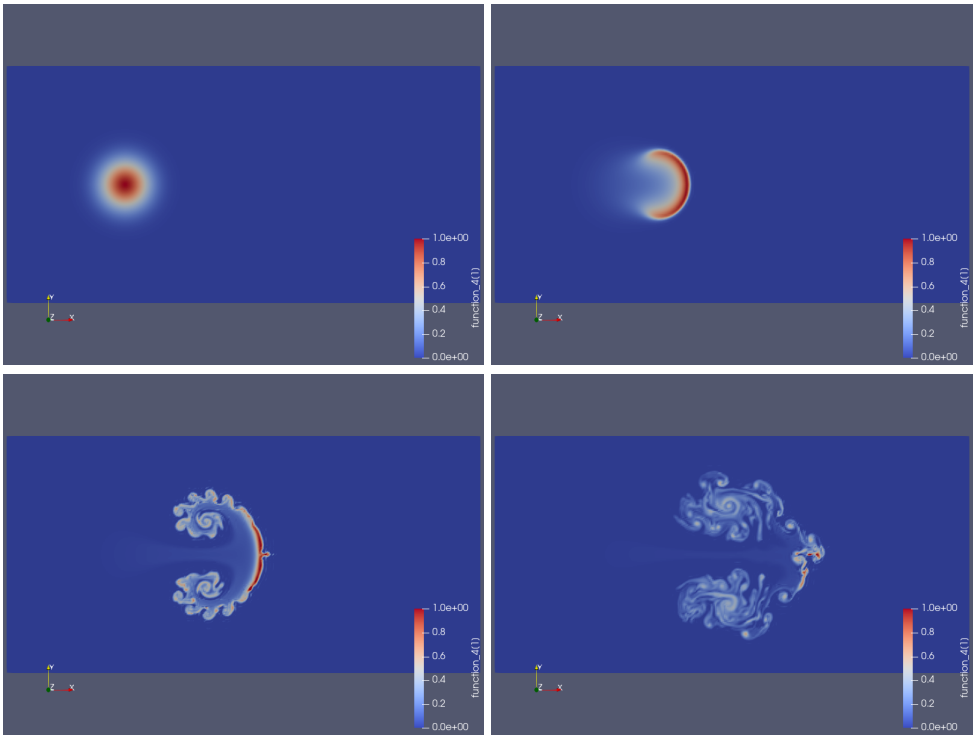


Figure 16: Numerical solution of the Hermes-3 Blob2D system, showing density, using a 64×32 mesh of square fourth-order DG elements, starting from a Gaussian density perturbation initial condition. Plots show density, left-to-right then top-to-bottom $t = 0, \frac{2}{3}, \frac{4}{3}, 2$. Generated by `Blob2D_explicit.py` [7].

to make the Poisson problem well-posed. The beams are given an initial density perturbation in the form of a sinusoid with one period across the domain.

The Vlasov-Poisson system is, for charged matter in a uniform neutralizing background,

$$\begin{aligned} \dot{f} + v\frac{\partial f}{\partial x} + E(x)\frac{\partial f}{\partial v} &= 0; \\ \frac{d^2\Phi(x)}{dx^2} = -\omega_P^2\left(\int_{-\infty}^{+\infty}\frac{f dv}{v_N} - 1\right); \\ E &= -\frac{d\Phi}{dx}; \end{aligned} \tag{5}$$

where the constant v_N is chosen such that the domain contains no net charge. The constant ω_p is the plasma frequency, which specifies the strength of the coupling and is the only parameter in the problem (excepting initial data). Physically this translates into $\propto e\sqrt{n}$ for electron number density n for spatially-homogeneous solutions. The function f is the distribution function (DF) for the charged particles, ρ is the charge density, and Φ is the electrostatic potential. Note that the earlier references contain a sign error in the RHS of the second equation (the physical effect of which is to make like charges attract, which is correct for the gravitational case, where of course the neutralizing background is unphysical, but not the electrostatic).

A brief note about the physics. These beams repel one another and so a concentration of charge density is associated with slower-moving particles. This is easily seen to be the case in the simulations.

Note that this problem is also the subject of the grantee proxyapp using the TensorRegions extension to the Nektar++ library, developed under grant T/AW084/21 which had the aim of exploring the use of Nektar++ for this type of phase-space fluid problem. The two-stream instability using TensorRegions suffers from two outstanding issues: firstly, the solution, while showing the correct general appearance, demonstrates a circular oscillation as if the centre of mass of the system is undergoing simple harmonic motion; secondly, there are visible artifacts associated to element edges that are not seen in other implementations. The aspiration of the current work is that it may give a way to diagnose and debug the Nektar++ code, e.g. if the same artifacts can be reproduced - this work is ongoing.

The system is implemented as an advection problem coupled to an elliptic solve. The elliptic solve is done outside of the time-stepping stages, which is expected to limit the accuracy of the time-stepping - as before, this can be remedied by constructing a DAE implementation. The evaluation of the right-hand side of the Poisson equation involves the integration of f over the velocity axis (this sort of thing is central to such kinetic problems and is technically a section of the radon transform of the phase space density) and this is accomplished using the R-space functionality of Firedrake (this means that a function - in this case the potential - in a two-dimensional `ExtrudedMesh` can be independent of one of the dimensions) and a projection of f into the R-space. Note that unfortunately this method currently works only in two-dimensions. The author is not certain that this projection mechanism is available for DG elements since an error is reported if trying to obtain the facet normals on an `ExtrudedMesh` and so a SU-stabilized CG implementation was used. The SU term leads to significant damping and stabilizes the long-term behaviour relative to the implementation described in [15], which showed energy gain over the course of the simulation.

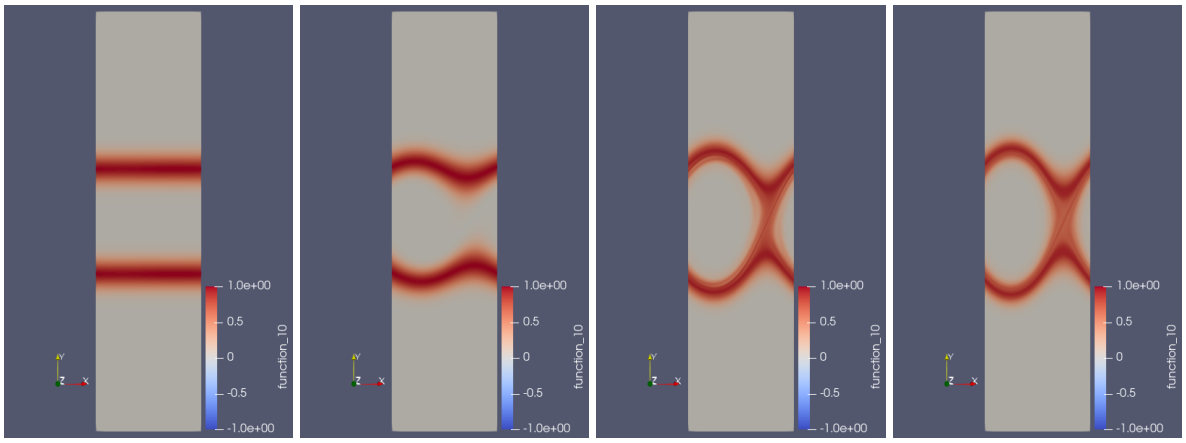


Figure 17: Numerical solution of Eqs.5 using a 64×256 mesh of square second-order CG elements with SU correction, starting from a bi-Gaussian-beam initial condition, and using $\omega_P^2 = 10$. Plots, L-R, are $t = 0, 8, 16, 24$, which can be compared to similar outputs in Fig.5 of [15]. Generated by `two-stream_instability_SU_repel.py` [7].

The UFL for the advection can be seen in Listing 11.

Listing 11: UFL code defining the equation system for the non-elliptic part of the Vlasov-Poisson system (`h` is the mesh cell size).

```
F = Dt(f)*v1*dx \
  - v1*div(f*driftvel)*dx \
  + 0.5*h*(dot(driftvel, grad(f)))*dot(driftvel, grad(v1))
        *(1/sqrt((driftvel[0])**2+(driftvel[1])**2))*dx
```

Note that this problem has interesting stability properties that were examined in Appendix A.3 of [14]. These properties could easily be studied further using Firedrake.

As well as the case of two beams of like-charged particles in a neutralizing background, it is possible to consider the situation where the beams have opposite charges i.e. they attract one another and no neutralizing background is needed. This is easily implemented by having two species (in the simple case presented here the two species have the same mass, but this is not necessary and stiffness can be introduced by making one species electrons and the other much heavier ions). The UFL for the advection in this case is shown in Listing 12.

Listing 12: UFL code defining the equation system for the Vlasov-Poisson system in the case of oppositely-charged matter beams.

```
F = Dt(f_a)*v1_a*dx + Dt(f_b)*v1_b*dx \
  - v1_a*div(f_a*driftvel_a)*dx - v1_b*div(f_b*driftvel_b)*dx\
  + 0.5*h*(dot(driftvel_a, grad(f_a)))*dot(driftvel_a, grad(v1_a))
        *(1/sqrt((driftvel_a[0])**2+(driftvel_a[1])**2))*dx \
  + 0.5*h*(dot(driftvel_b, grad(f_b)))*dot(driftvel_b, grad(v1_b))
        *(1/sqrt((driftvel_b[0])**2+(driftvel_b[1])**2))*dx
```

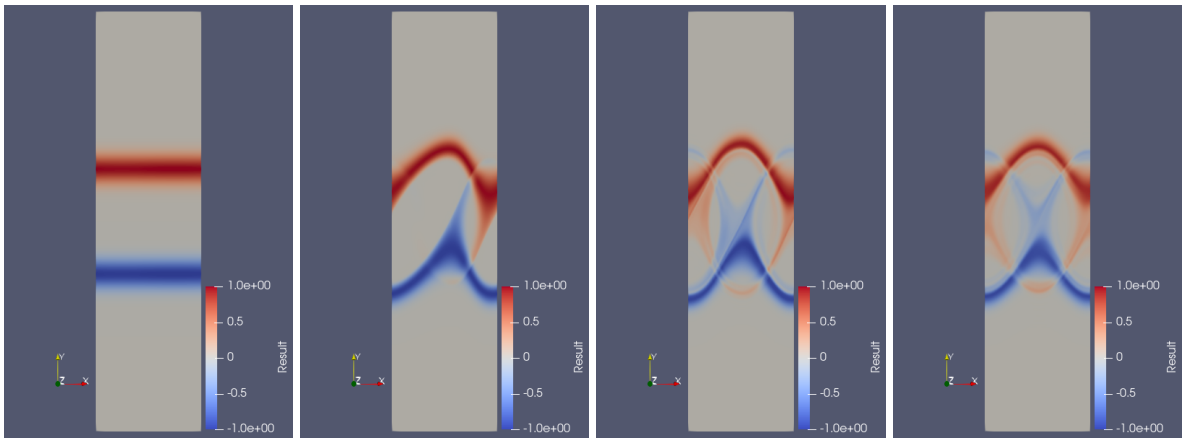


Figure 18: Numerical solution of Eqs.5 using a 64×256 mesh of square second-order CG elements with SU correction, starting from a bi-Gaussian-beam initial condition, and using $\omega_P^2 = 10$. Plots, L-R, are $t = 0, 8, 16, 24$. The plots are done by differencing the density of the two species using a ParaView `Calculator`; the plotted quantity then corresponds to the electrostatic charge density in phase space. Generated by `two-stream_instability_SU_attract.py` [7].

The potential is similar to that observed in the like-charges case - a near-sinusoid. This time the two species behave differently - because they feel opposite potential gradients, one species is fast at the same position that the other one is slow.

2.4.2 An analytic attractive equilbrium

Another system that can be tried in this solver is the analytic stationary solution found in Appendix A.1 of [16]. It must be noted that, due to the sign error mentioned earlier, this is not an electrostatic solution but rather one that applies in the case of a self-attractive force. Nevertheless, it can be studied numerically and has some value as a test case.

The solution, which satisfies Eqs.5 (but with the sign of ω_P^2 flipped), is

$$\begin{aligned} f(x,v) =& \frac{v_N}{\sqrt{2}\pi}\left(\frac{1-\frac{4\pi^2\Phi_0}{\omega_P^2}}{\sqrt{\Phi_0-\cos 2\pi x-\frac{1}{2}v^2}}+\frac{8\pi^2\sqrt{\Phi_0-\cos 2\pi x-\frac{1}{2}v^2}}{\omega_P^2}\right) \\ \rho(x) =& -\omega_P^2\left(\int f\frac{dv}{v_N}-1\right)=4\pi^2\cos 2\pi x; \\ \frac{d^2\Phi}{dx^2} =& -\rho. \end{aligned} \tag{6}$$

The non-singular case ($\Phi_0 = \frac{\omega_P^2}{4\pi^2}$) is

$$f(x,v) = \frac{4\sqrt{2}\pi v_N}{\omega_P^2}\sqrt{\frac{\omega_P^2}{4\pi^2}-\cos 2\pi x-\frac{1}{2}v^2}. \tag{7}$$

A quick test was done on this for $\omega_P^2 = 5\pi^2$ and the solution was time-evolved for 10 time units. Minimal change was seen in the form of the solution (Fig.19). Small perturbations were added to the analytic equilibrium and the solution was seen to relax back to something close to the analytic solution.

Solutions with the integrable singularity might also be tested - yet to be done.

3 A 3-D Hasegawa-Wakatani solver

Report [17] presented a NESO solver for the 2-D Hasegawa-Wakatani (hereafter HW) equations in a 3-D domain. This section describes how the 2D-in-3D Nektar++ EquationSystem has been extended to solve the full 3-D HW equations. These equations are of interest for two main reasons. Firstly, as a stepping stone on the path to modelling the LAPD problem, which is the near-term goal for plasma turbulence simulation in NESO, and second because they have already been studied using BOUT++ [18], offering an opportunity to verify the results with the help of grantees who are experienced users of that code.

The 3-D equations are similar to the 2-D form, but with the constant adiabacity parameter replaced by a second order spatial derivative in the parallel direction, that is:

$$\frac{\partial n}{\partial t} = -\left[\phi, n\right] - \alpha\nabla\cdot\left[\mathbf{b}(\partial_\parallel\phi - \partial_\parallel n)\right] - \kappa\frac{\partial\phi}{\partial y} \tag{8}$$

$$\frac{\partial \omega}{\partial t} = -\left[\phi, \omega\right] - \alpha\nabla\cdot\left[\mathbf{b}(\partial_\parallel\phi - \partial_\parallel n)\right] \tag{9}$$

Note that, as with the (modified) 2D Hasegawa Wakatani equations (see, e.g. [19]), n represents a fluctuations on a fixed background, often represented as $\hat{n}$. In the following discussion, we omit the 'hat' notation for brevity.

$\partial_\parallel$ indicates a spatial derivative parallel to the magnetic field, and $[a, b]$ is the Poisson bracket operator, defined as

$$[a, b] = \frac{\partial a}{\partial x}\frac{\partial b}{\partial y} - \frac{\partial a}{\partial y}\frac{\partial b}{\partial x}. \tag{10}$$

Note that, for a scalar field f

$$[\phi, f] \equiv \nabla\cdot(f\mathbf{v}_{\mathbf{E}\times\mathbf{B}}), \tag{11}$$

so in the code, the Poisson brackets are implemented via the Nektar advection API, having first calculated $\mathbf{v}_{\mathbf{E}\times\mathbf{B}}$ from Φ with a z-axis-aligned magnetic field.

As in the "2Din3DHW" system, Φ is updated at each stage of the Runge Kutta algorithm by solving a Poisson equation

$$\nabla_\perp^2\Phi = \omega \tag{12}$$

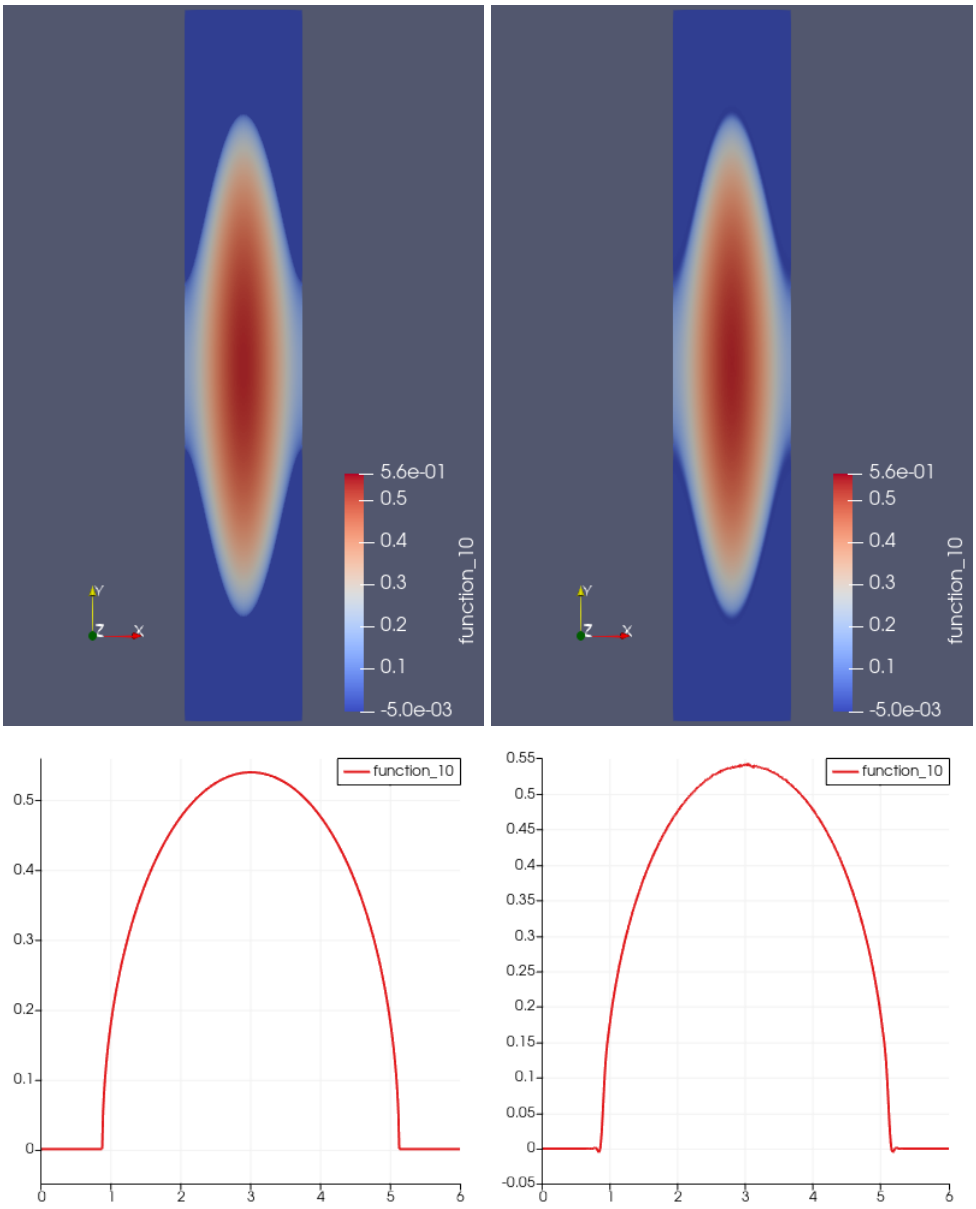


Figure 19: Numerical solution of phase-space density f from 7 using a 128×768 mesh of square second-order CG elements with SU correction, for $\omega_P^2 = 5\pi^2$. Left-hand plots show the initial data and right-hand plots the state after evolving for 10 time units; top is phase-space density profile and bottom is the profile at the central x-value of the upper plots. Note that the interaction can be seen to be attractive by the fact that greater particle speeds are associated to greater particle density. Generated by `Vlasov-Poisson_analytic_SU.py` [7].

where the $_\perp$ subscript signifies that the operator is restricted to the two-dimensional subspace transverse to the magnetic field.

Implementations of the 3-D HW equations in the literature, e.g. [20], typically include diffusion or hyper-diffusion (∇^4) terms in both the n and ω equations to add stability. In our solver this role is played, to some extent, by numerical diffusion associated with the discretisation.

3.1 Implementation details

The solver provides for two ways of specifying parameters in the Nektar++ configuration file. Users may either supply α and κ directly, or may set physical parameters which are then used to calculate α and κ when the Nektar EquationSystem is initialised.

In this latter mode, the code first computes the Coulomb logarithm for electron-ion interactions, $\log\Lambda_{e,i}$, based on equation 131 of [21], then the electron resistivity via the Spitzer formula with a singly charged ion, $\eta = 5.2 \times 10^{-5}\log\Lambda_{e,i}/(T_0^{3/2})$, [22]. α then follows from:

$$\alpha = \frac{T_0}{n_0 \eta e \omega_{ci}}, \tag{13}$$

where T_0 and n_0 are the background electron temperature in eV and number density in SI units, respectively.

Recall that the turbulent drive term coefficient, κ, controls the slope of the exponential background density profile. The user may set an approprate value for the scale length, λ_n and the code then computes κ, normalising to the ion gyro-radius $\rho_s = \sqrt{m_i T_0}/eB$.

$$\kappa = \frac{\rho_s}{\lambda_n}, \tag{14}$$

The parallel diffusion terms in Equations(8) and (9) are implemented via the Nektar++ diffusion API. A matrix $\mathcal{C}$ can be supplied such that, for a scalar field f, Nektar++ computes

$$\nabla \cdot (\mathcal{C} \nabla f) \ . \tag{15}$$

The matrix coefficients are labelled according to

$$\mathcal{C} = \begin{bmatrix} d_{00} & d_{01} & d_{02} \\ d_{01} & d_{11} & d_{12} \\ d_{02} & d_{12} & d_{22} \end{bmatrix}. \tag{16}$$

Setting $f = (\Phi - n)$, $d_{22} = 1$ and all other coefficients to zero, therefore yields the required term in the 3-D HW equations:

$$\nabla \cdot \left[\mathbf{b}(\partial_\parallel \phi - \partial_\parallel n) \right], \tag{17}$$

with $\mathbf{b} = (0, 0, 1)$.

As is the case for the "2Din3D" solver, n and ω are discretised using the Discontinuous Galerkin (DG) method, while the potential, Φ, is represented as a Continuous Galerkin (CG) field that is

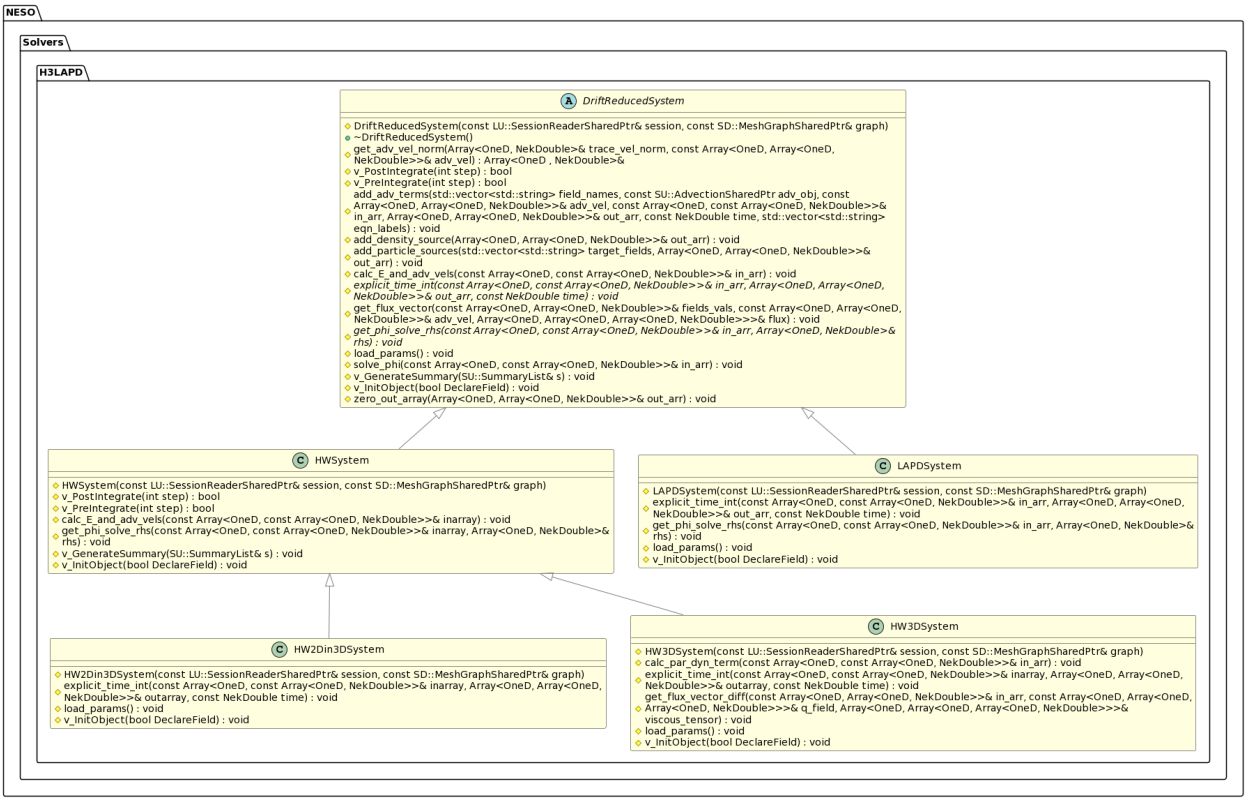


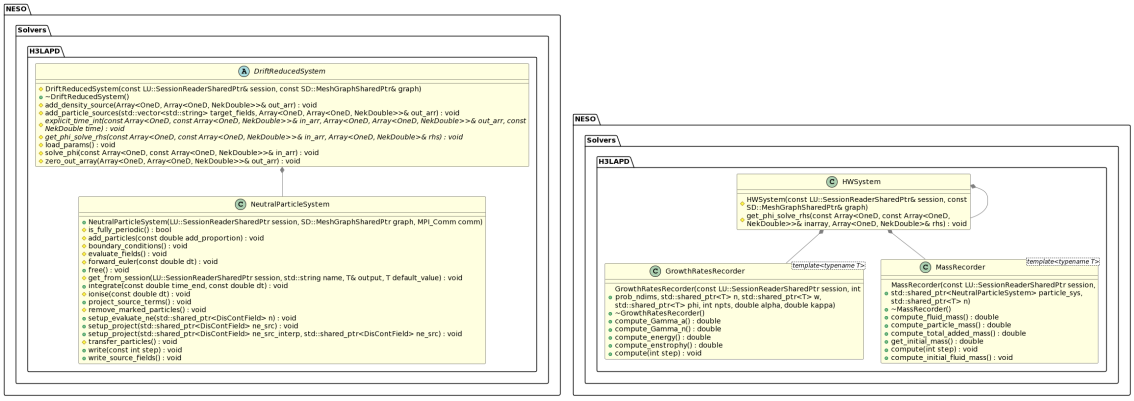
Figure 20: A PlantUML diagram showing the inheritance structure of EquationSystem classes in the H3LAPD (HERMES-3-LAPD) solver. Note that all member functions are shown, but member variables are excluded for the sake of brevity. Image generated from NESO header files using the hpp2plantuml [24] and plantuml [25] Python packages

recomputed from ω at each timestep. n and ω are evolved using an explicit 4^{th} order Runge Kutta scheme and the advection operations are stabilised using a first-order upwinding scheme.

Since several aspects of the numerical implementation described above are common to both the "2Din3D" and 3-D HW equation systems, substantial refactoring of the equation system classes (over and above that reported in M4c.3 [17]) was undertaken in order to avoid code duplication. The refactoring is also intended to properly encapsulate functionality relevant to equation systems that are in development, notably those used to solve the LAPD problem discussed in section 2.3.3 of report M4c.3. Figure 20 shows a PlantUML [23] diagram illustrating the new inheritance structure of C++ equation systems classes in the H3LAPD solver. Note that only member functions, and not member variables, are shown for the sake of brevity. Public function names are preceded by a green circle, protected functions by a yellow diamond, and private functions by a red square.

All of the systems inherit from an abstract base class called "DriftReducedSystem", which itself inherits from Nektar++'s "AdvectionSystem". This base class implements a number of pure virtual functions which all derived classes must implement, including:

- *explicit_time_int*: called by a Nektar time integrator (of explicit type) at each step to compute terms on the right-hand side of the time evolution equations.



(a) Neutral particle coupling (b) Solver diagnostic callback example

Figure 21: PlantUML diagrams showing the coupling between the base equation system class and the neutral particle system (a) and an example of attaching callbacks to a solver in order to run analysis or diagnostics (b).

- *get_phi_solve_rhs*: defines the right-hand side of the Poisson equation used to calculate the potential
- *load_params*: reads any class-specific parameters from the Nektar configuration file
- *v_InitObject*: a function inherited from the Nektar++ EquationSystem class that carries out class-specific initialisation at startup, including allocating temporary arrays, instantiating objects from the Nektar++ API, etc.

The base class also stores and initialises a system of neutral particles (see panel *a* of Figure 21), which can be configured, or disabled entirely, via parameters in the Nektar XML file. All derived classes then call functions inherited from the base class to interact with the underlying NESO-Particles objects. An example of a coupled simulation based on the 3-D HW solver, including ionisation and recombination, can be found in [26].

Two diagnostic classes are attached to the base class in order to monitor total fluid mass, particle mass, energy and enstrophy (panel *b* of Figure 21). These can be used for analysis but are also employed in integration tests of the solver hierarchy in order to verify that (e.g.) the process of ionisation conserves mass in coupled solvers and that the growth of energy and enstrophy matches analytically expected rates for the Hasegawa-Wakatani equations.

"HWSystem" includes data storage and implementation common to both the 2-D and 3-D versions of the Hasegawa-Wakatani equations, including variables to store the α and κ parameters (defined slightly differently in the two systems), an override of the *get_phi_solve_rhs* function, which retrieves the current values of the vorticity field at each quadrature point and an override of the *calc_E_and_adv_vels* function that passes values of the $\mathbf{v}_{\mathbf{E}\times\mathbf{B}}$ (calculated in the base class implementation) to the electron advection object.

Finally, the "HW2Din3DSystem" and "HW3DSystem" classes implement different versions of *explicit_time_int*, adding the appropriate terms to the n and ω equations in each case.

3.2 Results and discussion

The results described in this section are from a simulation run in a cuboid-shaped domain using a 5x5x10 mesh with unit cube elements. Element basis functions were set to be modified 7^{th} order Legendre polynomials of the form described by Karniadakis and Sherwin [27]. The parameters α and κ were set to 0.1 and 3.5 respectively. Note that this configuration does not correspond to a specific physical scenario of interest, but was instead chosen to demonstrate evolution to a turbulent state and to test the operation of the new equation system in the solver. The initial conditions were set to be Gaussian in the perpendicular direction, with a width of 0.5 normalised units, and sinusoidal in the parallel direction with a period one fifth of the box length. Periodic boundary conditions were used in all three dimensions.

The simulation is set up as a NESO solver example, stored in the GitHub repository [28] at `examples/H3LAPD/3D-hw`, at time of writing. The example directory contains a Nektar++ configuration file and a GMSH file that defines the characteristics of the mesh. Utility scripts are provided to generate a Nektar++-compatible mesh from the GMSH file and then run the example in a separate directory, using MPI.

3.2.1 Density evolution

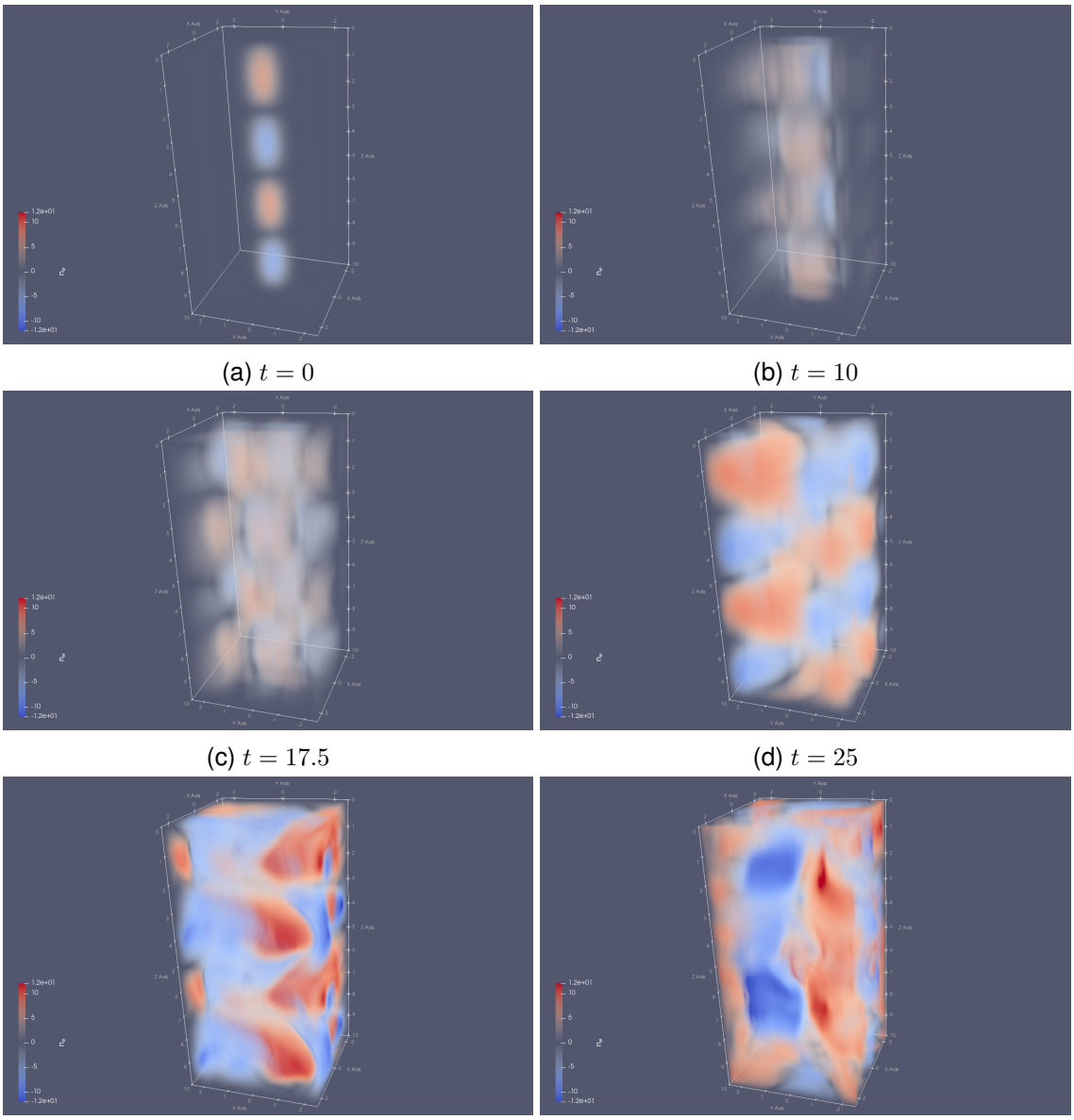
Figure 22 shows the evolution of the density field captured at six output times. In general, one might expect the relatively high value of κ to drive linear growth of perturbations via the drift-wave instability, which then undergo secondary instabilities that drive non-linear growth, generating shear and giving rise to zonal flows. While some evidence of turbulent structure is apparent in the final output, at t=40, it appears to be somewhat less prevalent than in simulations run with the '2Din3D' implementation of the solver (c.f. Figure 6 in [17]), presumably due to the addition of the diffusive term in the field-parallel direction. Examples of similar simulations in the literature (e.g. [19]) show that, in general, a steady state is reached in which the resistive dissipation (controlled by α) balances the turbulent drive term. Note, however, that results in Section 3.2.2 suggest that such a state has not yet been reached by t=40.

3.2.2 Energy and enstrophy

Two quantities of interest in HW simulations are the total system energy and enstrophy. These can be calculated with integrals over the domain:

$$E = \frac{1}{2}\int (n^2 + |\nabla_\perp \phi|^2)\ \mathbf{dx} \tag{18}$$

$$W = \frac{1}{2}\int (n - \zeta)^2\ \mathbf{dx} \tag{19}$$



(a) $t = 0$ (b) $t = 10$

(c) $t = 17.5$ (d) $t = 25$

(e) $t = 32.5$ (f) $t = 40$

Figure 22: Evolution of the electron density in the 3-D Hasegawa-Wakatani solver at several output times.

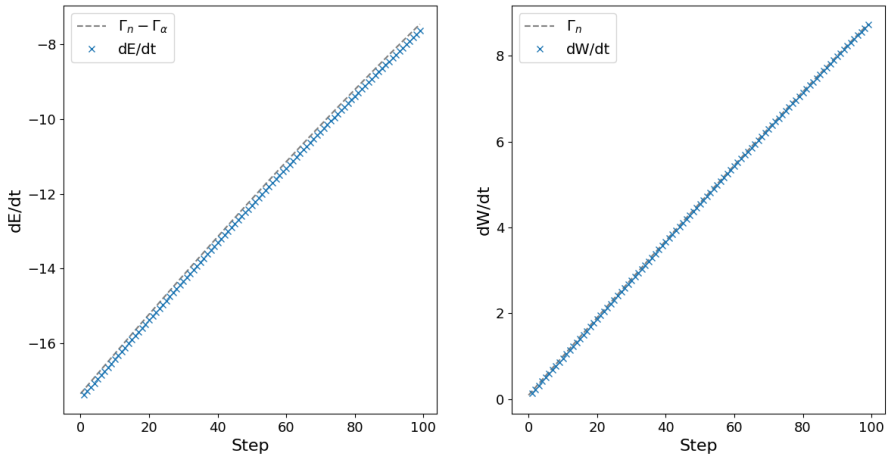


Figure 23: Rates of energy and enstrophy growth, in normalised units, for the first 100 steps of the simulation. Dashed lines indicate analytic expected values in each case, calculated based on Equations(20) and (21).

As explained in reference [19], the energy and enstrophy are expected to evolve at rates

$$\frac{dE}{dt} = \Gamma_n - \Gamma_\alpha \tag{20}$$

$$\frac{dW}{dt} = \Gamma_n \tag{21}$$

where

$$\Gamma_\alpha = \int \alpha[\frac{\partial}{\partial z}(n-\phi)]^2 \, \mathbf{dx} \tag{22}$$

$$\Gamma_n = -\kappa \int n \frac{\partial \phi}{\partial y} \, \mathbf{dx} \tag{23}$$

In order to study these quantities with NESO, a solver callback was added to the HWSystem class to compute E, W, Γ_α and Γ_n at each step and write them to file.

Figure 23 shows the rates of change of E and W over the first 100 steps of the simulation (in the linear growth phase), together with the expected values calculated from Equations(20) and (21). The former are calculated by differencing each quantity between consecutive steps and dividing by the system timestep. Both quantities grow at a rate very close to the expected values, verifying, to a degree the implementation of the “3DHW” equation system.

In Figure 24, the evolution of the total energy is plotted over the full simulation duration. In a steady state, where resistive dissipation matches turbulent drive from the background density profile, one expects $\Gamma_\alpha = \Gamma_n$, such that the total energy saturates. In this simulation it is clear that the system is still in flux at $t = 40$ and has not yet reached the saturation point.

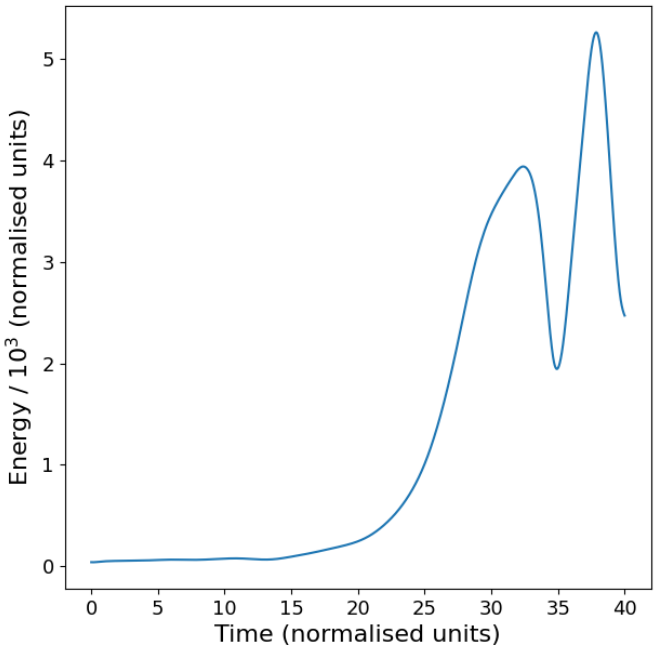


Figure 24: Evolution of the total system energy over the full duration of the simulation. Time and energy are plotted in normalised units.

3.3 Summary

The 3-D Hasegawa-Wakatani equations have been implemented in NESO by adding a new Nektar++ equation system to the "H3LAPD" solver. Substantial refactoring work has been undertaken to avoid code duplication and to facilitate coupling between the various H3LAPD systems and an ensemble of kinetic neutrals modelled with NESO-Particles, an example of which is demonstrated in report [26].

Analysis of the total energy and enstrophy in our example simulation demonstrates good agreement with the evolution predicted analytically, and reveals that the total duration may be slightly too short to reach a steady state. Further verification of our 3-D HW implementation is already underway via a comparison to the results presented by the York grantees in report [29].

4 Producing Field-Aligned Meshes

This section describes the work done to produce field-aligned meshes for calculating anisotropic diffusion.

In finite difference (and low-order finite-element) codes it has been found that anisotropic diffusion can only be solved accurately if the mesh is aligned with the magnetic field. Otherwise the solver will converge to the wrong solution. This issue may be less significant for higher-order finite difference basis functions, but there could still be performance benefits if a lower-order basis could be used to achieve the same level of accuracy.

While it is possible to generate meshes where elements follow magnetic field-lines, the problem that arises is that, in general, field lines will not form a closed loop after a single circuit of a tokamak. This means there would be a discontinuity at $\phi = 0, 2\pi$ where faces of adjacent elements would not align. In finite difference codes the toroidal direction is not actually aligned with the field but interpolation is used when taking derivatives along the field-lines.

A similar approach can be used for finite element methods. First, a poloidal mesh is generated in the plane with $\phi = 0$. The field line intersecting each control point in that mesh is then followed to $\pm\Delta\phi/2$, where $\Delta\phi$ is the toroidal grid spacing. These field lines will make up the edges of one "layer" of elements. Successive layers will be identical except rotated a suitable angle in the toroidal direction. In general the poloidal faces of elements in adjacent layers will not align and are said to be "nonconformal". There are established algorithms to interpolate values from nonconformal elements when using the discontinuous Galerkin method[30]. DG is less efficient than Continuous Galerkin for elliptical problems such as diffusion. In production, however, there will also be hyperbolic terms in the equations, at which point it will be necessary to use DG anyway.

Note that it is assumed that the equilibrium magnetic field is axisymmetric, i.e., does not depend on ϕ. This means that the shape of elements only needs to be computed for one layer; all layers would be the same except rotated in ϕ. The techniques outlined in this section could be modified to work without axisemmetry but this would be more computationally expensive as every layer would need to be calculated individually.

4.1 NESO-fame

It is necessary to generate a mesh that can be used by Nektar++. Gmsh meshes can be converted to the Nektar++ format and, if necessary, undergo further manipulations using the NekMesh tool. This is the workflow which is normally used. Unfortunately, Gmsh and most other existing software for generating meshes does not allow one to specify the exact arrangement and alignment of elements within the domain. This meant it was not suitable to ensure the elements are field-aligned. Instead, new software had to be written. This was called NESO-fame, where "fame" stands for "field-aligned mesh extrusion".

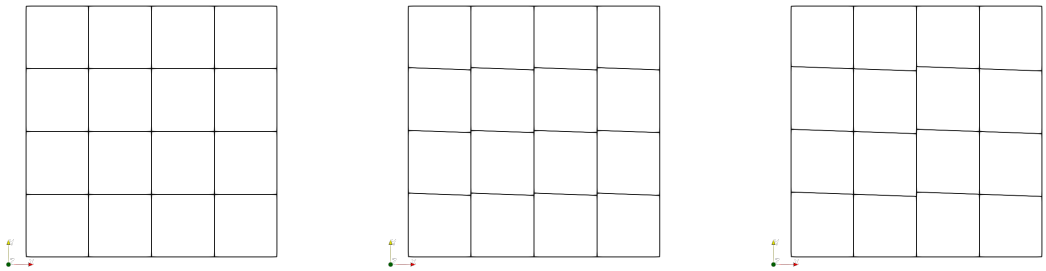
Data on equilibrium magnetic fields for tokamaks are usually distributed using the G-EQDSK format. To avoid needing to implement a new reader for EQDSK files, the existing `hypnotoad` package was used. This software was designed to generates meshes for the BOUT++ finite difference library[31]. In addition to reading the EQDSK file, it can be used to generate (most of) the 2D poloidal mesh from which the 3D mesh will be extruded. The only disadvantage of using `hypnotoad` is that it is released under the GNU General Public License, meaning that NESO-fame also had to use this license. However, NESO-fame is sufficiently ancillary to the main NEPTUNE codebase that this was considered an acceptable compromise. Furthermore, it remains possible to write a replacement for the `hypnotoad` functions used in fame in the future, if it becomes necessary to distribute the latter under a more permissive license.

`hypnotoad` is written in Python, so the NekPy Python bindings were used to produce Nektar++ meshes from it. This required a few additional class and method bindings to be added to NekPy, particularly for nonconformal interfaces.

4.2 2D and Simple 3D Meshes

Initial work focused on very simple 2D and 3D meshes. For further simplicity, these used Cartesian coordinates rather than cylindrical. NESO-fame would first generate a structured 1D or a 2D mesh. It would take the angle between the magnetic field and the normal direction of this mesh and extrude each element along that angle. The user would specify how many "layers" there would be in the direction of the field. Between each layer is a nonconformal interface (see Figures 25b and 26b). Each layer can be further subdivided into conformal elements in the field direction, if desired (see Figures 25c and 26c). This makes it possible to experiment to determine what frequency of conformal and nonconformal interfaces can give the best accuracy for the least computational cost.

It was planned to use the existing Nektar++ anisotropic diffusion mini-app [32] to compare the accuracy of 2D meshes with no field alignment, with fully-conformal field alignment, and with field alignment and non-conformal interfaces. The computational cost of each of these would also be compared. However, it was discovered that the mini-app only worked with the Continuous Galerkin method and that Nektar++ did not support Neumann boundary conditions with the Discontinuous Galerkin method. These only became available in the final weeks of the NEPTUNE project. Support for 3D meshes in the mini-app was not available at the time of writing.



(a) Unaligned, Conformal (b) Aligned, Nonconformal (c) Aligned, Subdivided

Figure 25: Examples of various types of 2D meshes which are aligned (or not) to a magnetic field with a 2° angle to the x-axis.

4.3 Generating a Poloidal Mesh

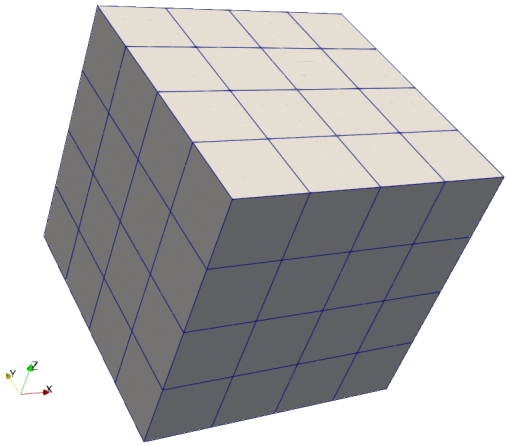
While awaiting the changes to Nektar++ necessary to benchmark anisotropic diffusion in the 2D case, work continued to generate realistic tokamak meshes. The first step of this was to produce a useful 2D poloidal mesh.

The starting point was a poloidal mesh generated by `hypnotoad` from a G-EQDSK file. `hypnotoad` provides a large number of configurable parameters and a graphical user interface which allows one to experiment with the values of these and preview the result. These meshes are made up of quads. Two edges of each quad follow flux surfaces and the other two are (by default) perpendicular to flux surfaces. This mesh can be brought arbitrarily close to the core of the tokamak, but will always have a hole in the centre. The mesh will not conform to the wall of the tokamak (the shape of which will typically be included in a G-EQDSK file). Some portions of it may protrude past the wall, while in other regions there may be a gap between the mesh and the wall; compare Figures 27a and 27b.

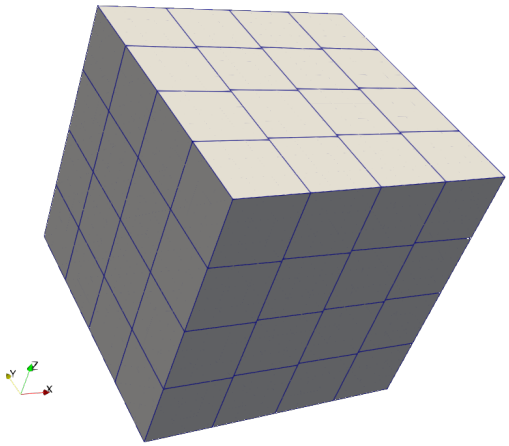
4.4 Filling the Gaps

Filling the hole in the centre was straightforward. The quad edges closest to the centre are used as one edge of a triangular element, with the other edges formed by perpendicular segments connecting to the O-point. Due to numerical difficulties, these edges deviate slightly from perpendicular when very near the centre, instead taking the shortest path in Euclidean space for the rest of the distance. Note that extending the mesh into the centre is optional; if this region is not of interest then a boundary condition can be imposed on the inner edge of the annulus instead.

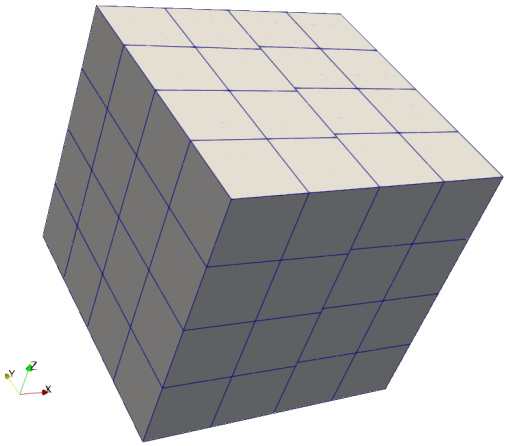
Next, any vertices of the `hypnotoad` mesh which lie outside the tokamak wall are eliminated. The user may also specify some minimum distance between these vertices and the wall; any vertices nearer than that will also be removed. It is necessary to follow the field lines starting at these vertices (see next section), to ensure that at no point will the edge of the element extruded from this node pass outside the wall or within the minimum distance to the wall. The `meshpy` library, with



(a) Unaligned, Conformal

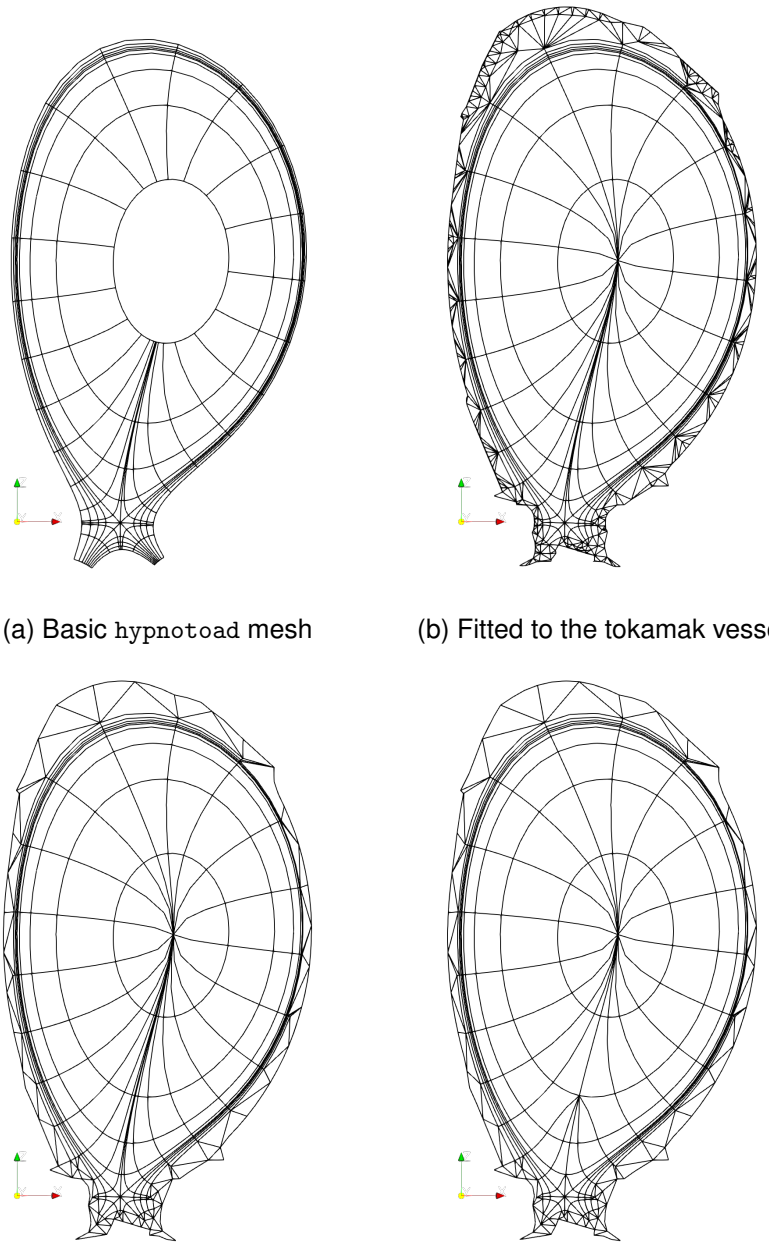


(b) Aligned, Nonconformal



(c) Aligned, Subdivided

Figure 26: Examples of various types of 3D meshes which are aligned (or not) to a magnetic field with a 2° angle from the x-axis towards the y-axis and a 2° angle from the x-axis towards the z-axis.



(a) Basic `hypnotoad` mesh

(b) Fitted to the tokamak vessel

(c) Adjusted wall resolution

(d) Merged very narrow elements

Figure 27: Poloidal meshes from which 3D meshes can be extruded, demonstrating how they are adjusted from the output of `hypnotoad`. (a) The mesh generated by `hypnotoad`. (b) The previous mesh but restricted to inside the tokamak vessel, with any empty spaces filled in with triangles. (c) The same, except with the wall resolution adjusted to more closely match that of the main mesh. (d) Very narrow elements above the x-point have been merged, so this region is not over-resolved.

the `triangle` code as a backend, was then used to produce triangles to fill the space between the remaining `hypnotoad` vertices and the tokamak wall.

An example of applying these adjustments can be found in Figure 27b.

4.4.1 The Wall

The information on the shape of the wall provided in the G-EQDSK data consists of a series of linear segments. The length of these segments can vary significantly and may be much smaller than the resolution of the mesh being generated. Generating a high quality mesh therefore requires adjusting the shape and resolution of the wall, before generating the triangular elements between it and the `hypnotoad` mesh. The user may specify a target wall resolution. This is expressed as a multiple of the average size of the `hypnotoad` mesh edges that are closest to the wall. The user will also specify the minimum size of any sharp angles which should be preserved after regridding.

Adjacent line segments are put into groups, separated by sharp angles above the user-provided threshold. The total length of each group in space is calculated. If a group has a length less than 0.1 of the target size then it is deleted and the ends of adjacent groups moved to its centre. Cubic splines are used to interpolate the shape of the remaining groups. A number of curved segments, as close to the target length as possible, will be created for each group. The length of the segments created from the same group will be equal, but in general the length of segments from different groups will not be. An example of a mesh with a regridded wall can be found in Figure 27c.

4.4.2 Improving Mesh Quality

All of these 2D elements may be higher-order (i.e., have non-linear edges). This allows the flux surfaces and perpendiculars to be followed more accurately. The triangles near the wall of the tokamak will mostly have straight edges, except where they adjoin the `hypnotoad`-generated mesh or the wall itself. As the `triangle` code assumes first-order elements it can sometimes produce shapes that are self-intersecting once curved edges are applied. To guard against this, the Jacobian of each triangle adjacent to the wall is checked. Those with negative Jacobians (indicating self-intersection) may be merged with an adjacent triangle to produce a quad or, if no such quad is well-formed, forced to be first-order.

Lines that are perpendicular to the flux surfaces tend to diverge near the X-point. Ensuring there is adequate resolution in that region will often result in overly-resolved areas towards the O-point, where these lines converge. Columns of elements radiating away from the X-point and towards the O-point are checked to see if elements become extremely narrow. If the ratio between their length and width exceeds some threshold (set by the user), they are merged with adjacent elements. This means computational resources will not be wasted with unnecessary resolution. An example of a mesh to which this process has been applied can be found by comparing Figure 27c to Figure 27d.

4.5 Following the Field

The path of a field line is the solution to the following system of differential equations:

$$\begin{aligned} \frac{dR}{d\phi} &= \frac{B_R(R,Z)}{B_\phi(R,Z)}, \\ \frac{dZ}{d\phi} &= \frac{B_Z(R,Z)}{B_\phi(R,Z)}. \end{aligned} \tag{24}$$

A solution would have the form $R(R_0, Z_0, \phi), Z(R_0, Z_0, \phi)$, where R_0 and Z_0 are the coordinates where the field line intersects the poloidal plane at $\phi = 0$. In general there will not be a closed form for this solution and it must be integrated numerically from the G-EQDSK equilibrium data. To extrude the 2D elements into the toroidal direction, this integral is evaluated starting at the control points on each of the edges of the toroidal mesh. The integral is taken from $\phi = 0$ to $\phi = \pm\Delta\phi/2$, where $\Delta\phi$ is the spacing of elements in the toroidal direction. The distance travelled along the field line is also calculated. The results for each field line are then interpolated against the normalised distance travelled, to ensure that any new control points are equally-spaced on the line.

This will not work for elements falling between the mesh generated by `hypnotoad` and the tokamak wall, as the field line may leave the tokamak vessel. The edges of these elements are therefore left unaligned, with $R(\phi) = R_0$ and $Z(\phi) = Z_0$. Elements immediately outside the `hypnotoad` mesh will have some edges that are field-aligned and some that are not. This can sometimes cause the element to be self-intersecting. It is also anticipated that the sudden transition might cause numerical artefacts. To avoid this, the user can specify the number of steps to take between elements that are fully field-aligned and fully unaligned. Elements within the `hypnotoad` mesh but near its edge will have edges that are the weighted average between the aligned and unaligned shape, with the weight of the unaligned component rising as the edge is approached. It is this weighted average that is used when evaluating whether a vertex and its corresponding field line lie within the tokamak vessel (see Section 4.4).

An example of the meshes generated by this process can be found in Figure 28.

5 Grantee Deliverable Summaries

Each section below summarises the deliverables received from a grantee, with one report or code deliverable per subsection. A link is provided in the title to the location of the accepted report, where applicable. Accepted reports are held on the Documents repository of the ExCALIBUR-NEPTUNE organisation on GitHub, to which belong other repos containing software developed under project NEPTUNE. Note that some of these repos may be access-controlled, please email `neptune@ukaea.uk` if difficulties are encountered in seeing the material.

5.1 Reports received under Grant T/AW086/22, PO 2070839

The Oxford grantee produced the following reports

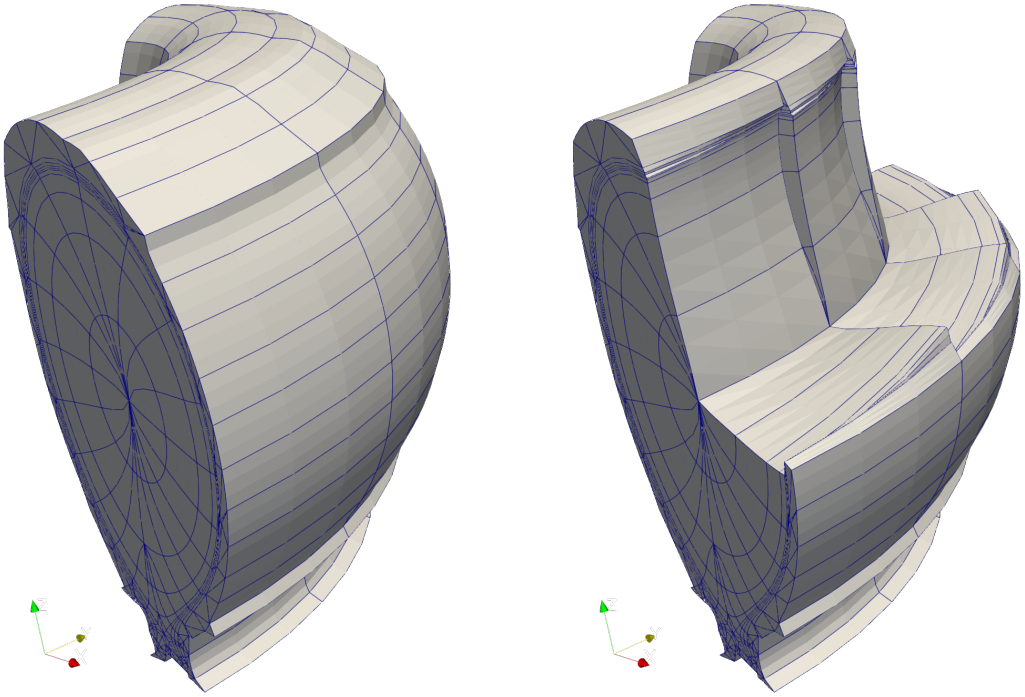


Figure 28: The field-aligned mesh for a quadrant of a tokamak. On the left is the full mesh, while on the right some of the elements have been hidden to give a view of the mesh interior. Edges and quads on the surface of the tokamak are not field aligned and are therefore conformal. This contrasts with the internal elements.

5.1.1 Report 2070839-TN-03 [33]

The report examines how best to formulate the calculation of the electrostatic potential ϕ when solving the drift kinetic equations, to ensure the potential is accurately and stably computed numerically. Use of the Boltzmann response to calculate ϕ may give rise to a significant error. The first requirement is therefore a way of deciding when the Boltzmann response is inadequate, which is addressed by a theory that makes a perturbation to the Boltzmann response in the estimation. If the perturbation is small, then it is used to correct the Boltzmann response, otherwise a far more complicated formulation leading to the electron fluid equations for electron density, velocity and temperature is argued to be necessary. One truncated variant is shown to be equivalent to drift-reduced Braginskii. The analytic results in the report should be valuable for ensuring accurate and efficient numerical solution of drift kinetic equations.

5.1.2 Report 2070839-TN-04 [34]

The report describes work designed to test using the Method of Manufactured Solutions (MMS), correct implementation of the 2-D drift kinetic model introduced in previous reports from Oxford, and restated in the current report. This work has exposed limitations both of high order finite elements and of MMS, one of the latter being somewhat subtle. Dealing first with the limitations of high order, the peculiar nature of the boundary condition implies that the chosen solution becomes zero at off-node points, ie. effectively discontinuous. Moreover, solutions corresponding to realistic source terms are expected to have the sheath singularity dependence, $\phi \propto \sqrt{z}$, where ϕ is the electric potential. Spectral convergence is only to be expected when approximating a smooth solution. Otherwise, two strategies to obtain better convergence are possible, the first (adopted) is to use lower order elements, but more of them, and this was successful. The second option, particularly applicable to a known form of singularity, is to use singular elements, treated in detail by Strang and Fix [35], as referenced by Boyd [36, § 16.6]. This suggestion is added to the list of options for further exploration and testing of the model implementation at higher accuracy.

The first limitation of MMS exposed by the study is inappropriateness of the function chosen, which does not have the expected singular potential behaviour. Secondly, evidence is presented of the more subtle issue that the source implied by the solution chosen tends to provoke a short wavelength instability. Convergence is restored by appropriately chosen numerical dissipation, again of course at the expense of accuracy. Nonetheless, it is hard to doubt the correctness of coding and likely fidelity of simulation to be achieved using the program, for which a link to the repository is given in Appendix A.

5.1.3 Report 2070839-TN-05 [37]

Report 2070839–TN–05 continues the series of high-quality reports produced by Oxford for the project, with clear explanations, notably as to why particular options have been selected. The report concentrates on the treatment of the electrons, leaving description of the kinetic modelling of the ion and neutral species to previous reports. A further assumption, made for the purposes of the Oxford development, is that not only are the models 1-D in space, but also in velocity

space $1d1v$, where the relevant component is parallel to the magnetic field, which is here taken to be uniform and straight in the z-direction.

The electron treatment is moment-based, implying a set of fluid-like equations for mass/charge, momentum and energy. The electron momentum balance gives the electric field which can otherwise be hard to calculate. Boundary conditions are chosen so that the electrons have the same density and velocity as the ions. Tests show that making zero flux assumptions, an electron Boltzmann response is recoverable, and that using the Braginskii closure a steady state is achievable, although the E-field is noisy.

5.1.4 Report 2070839-TN-06 [38]

The report 2070839–TN–06 continues the series of high-quality reports produced by the Oxford grantees for the project, with clear explanations, notably as to why particular options have been selected. The model is $1D2V$ for the ions, which are coupled to Boltzmann electrons as in previous Oxford reports, the novelty here being the collision operator for the ions. Two different operators - the simple Krook and a more complicated gyro-averaged Fokker-Planck type - are introduced as indicated in the report title.

The simple Krook operator is used in the range of Knudsen numbers 0.1 to 100 where both accurate modelling and intuition about likely results are hard to come by. Nonetheless, there is a satisfactory test of the code by the Method of Manufactured Solutions (MMS), and physically plausible 'smooth' solutions are found for a Maxwellian particle source, in the presence of a small artificial viscosity. There are numerical challenges associated with the latter Fokker-Planck operator since it leads to fourth order derivatives, and its calculation involves integration over the infinite range and an inverse square-root singularity at one integration limit. Progress has been made in the understanding of these issues.

5.1.5 Report 2070839-TN-07 [39]

The report 2070839–TN–07 describes in sufficient detail for implementation, the use of a high order basis to treat the Rosenbluth-MacDonald-Judd form of the Fokker-Planck collision operator for charged particle species. The basis consists of the direct product of Lagrange polynomials in velocity coordinatesr $v_{\|}$ and $v_{\perp}$ defined on Lobatto points except for the element including $v_{\perp} = 0$. Since the operator involved is both nonlinear and integro-differential, and there are challenging boundary conditions at infinity, as well as the usual issues expected at the origin of a radial coordinate $v_{\perp}$, the analysis involves approximately 100 numbered equations. Preliminary testing and timing of the initialisation and a single step calculation using the operator is successfully performed, with results that the subsequent discussion concludes are encouraging for further implementation.

5.1.6 Report 2070839-TN-08 [40]

The report 2070839–TN–08 describes openly and honestly progress made in pursuit of a challenging technical objective. The objective of a more accurate electron distribution function is pursued further in report 2070839–TN–10, of which see the summary below for more technical details of the objective and the latest progress.

5.1.7 Report 2070839-TN-09 [41]

The report 2070839–TN–09 describes the introduction into the ExCALIBUR drift-kinetic ("Oxford") model of a magnetic field of spatially varying *strength* (formerly only field direction varied). This z-dependence of B_z leads to more complex ion dynamics, whilst the electrons are treated as adiabatic. It is argued that in the 1-D and 2-D situations considered, it is adequate to demonstrate magnetic mirroring without forcing the field to be solenoidal. The Method of Manufactured Solutions is used to test the resulting software, demonstrating spectral convergence in both one- and two- space dimensions, where in the latter case the solution is assumed to be periodic in the radial coordinate. The success of the tests bodes well for more complicated open field geometries.

5.1.8 Report 2070839-TN-10 [42]

The report 2070839–TN–10 returns to a problem attacked unsuccessfully in previous work [43], viz. to find an electron particle distribution function f_e by replacing use of the Boltzmann assumption with a less prescriptive formulation. Indeed, the new model for evolution of f_e is almost as detailed as for the ion species f_i, the major difference being the assumption that f_e may be evolved in pseudo-time to find a steady-state. The new model equation (42) for $g_e \propto f_e$ is inevitably of a complicated integro-differential form with boundary conditions that are linked to the f_i distribution via the electric potential. The boundary conditions are reformulated to impose a zero parallel current.

Following implementation of the new boundary conditions and other changes to the coding, Computations assuming up-down spatial symmetry obtain physically reasonable f_e except very near the domain boundaries where a finite grid-scale oscillation is observed. Thus, the work demonstrates significant progress towards more accurate calculation of electron behaviour, but indicates that more research is still needed.

5.2 Reports and code deliverables received under Grant T/AW085/22, PO 2068435

The KCL grantee produced the following reports and code deliverables

5.2.1 Report 2068435-TN-01 [44]

The report 2068435-TN-01 presents in Section 2, Nektar++ implementations of three HERMES-3 examples in 2-D, viz. `Blob2D`, `Blob2D-Te-Ti`, and `Blob2D-Te-Ti-Turb`, the last of these in preliminary form. All have domains that are simple square regions, but the equations become more numerous and complicated with each example. Section 3 contains an explanation of the various Nektar++ components needed to implement a Discontinuous Galerkin (DG) solver and their use in the API. Section 4.1 contains implementation details including two alternative ways of solving the Poisson problem in the `Blob2D-Te-Ti` case; one method works while the other does not. Further work is needed to understand the reason for this, possibly there are problems with the dimensionless scaling. Section 4.2 considers an issue with the initial version of the `Blob2D` proxyapp, in that the numerical flux implementation in the DG solver was not fully conservative, however little difference is observed after modification. Section 5 suggests further work, incidentally mentioning changes to help developers deal with multi-species problems.

5.2.2 Report 2068435-TN-02 [45]

The report 2068435-TN-02 discusses improvements made to the Nektar++ mesh generation toolkit that cover many different aspects. It contains links to the relevant merge requests in the Nektar++ repository for the new and corrected code.

The report describes the following:

1. Completion of work on r-adaptation of mesh (ie. the movement of finite elements according to suitable criteria, without introduction of additional elements), with an associated figure showing anisotropic refinement of a grid around a circle.
2. Usability improvements with relation to mesh quality - statistics of the values of mesh-map Jacobians are printed out to the screen including graphics showing the distribution of Jacobians.
3. Bug fixes and improvements for domains incorporating periodicity have made periodic domains with pyramidal cells more robust, which should improve the situation where oscillations would emanate from periodic boundaries.
4. Improvements to isoparametric layer mesh generation in boundary layers. Previously, face curvature was not used, which led to low quality meshes especially at low resolution. This has now been fixed.
5. An examination of tokamak-relevant first wall geometries, which discusses the generation of a higher order mesh from simplified CAD models of a divertor concept. The bad elements which were found, are attributed to areas of poor CAD parameterisation.
6. Some future directions for NekMesh post-NEPTUNE. This section describes planned (and funded) work to make the NekMesh code more robust, extend the Python API, and to provide a standalone GUI for use by non-experts. Another workstream is the development of an Industrial Pipeline and wrapping that functionality back into NekMesh and Nektar++. Finally,

plans are laid out to incorporate NekMesh into Nektar++ so that the mesh can be modified at runtime, enabling eg. dynamic load balancing.

5.2.3 Code deliverable 2068435 Deliverable 4, Task 3.2

This deliverable consists of modifications to the Nektar++ codebase, submitted in the form of merge request !1759 in the main Nektar GitLab repository. It entails the addition of the ”MUI” (Multiscale Universal Interface) library Nektar++ as an alternative coupling framework to ”CWIPI”.

MUI has been enabled via additions to the MPI communications code, as is required for coupling frameworks over a split MPI commicator. Further alterations have been made to the simulation configuration tools, so that coupling may be defined via the standard user interface. In addition to these source code changes, there is a DummyMui EquationSystem, which serves as an example of how to couple codes (in this case two solvers within Nektar++) in 1, 2 and 3 spatial dimensions.

All tests added to the code as part of the merge request have been confirmed to execute successfully by a UKAEA reviewer.

5.2.4 Code deliverable 2068435 Deliverable 5, Task 3.3

This deliverable consists of modifications to the Nektar++ codebase, submitted in the form of two merge requests, !1379 and !1752 in the main Nektar++ GitLab repository. They expand Nektar’s existing set of Python bindings (NekPy) to include the SolverUtils library, which allows time-evolving Nektar solvers to be run directly from Python for the first time.

Demos are provided for an unsteady implicit diffusion solver and a Helmholtz solver, both in 2D domains.

In the context of project NEPTUNE, these changes have two important benefits:

1. The potential for rapid prototyping of new equation systems in Python, reducing overall development time
2. The removal of one of the major obstacles to building a Python DSL for Nektar++ solvers

All tests added to the code as part of the two merge requests have been confirmed to execute successfully by a UKAEA reviewer.

5.2.5 Code deliverable 2068435 Deliverable 1, Task 1.3

This deliverable consists of modifications to the Nektar++ codebase, submitted in the form of merge request !1743 in the main Nektar GitLab repository. It is the third and final part of deliverable 1, which comprises a significant refactoring of Nektar’s core libraries, with the aim of enhancing the modularity and performance portability of the code.

The changes that have been made decouple operators from the data that they act upon, lifting a restriction that was significantly limiting the capacity of the code to be performance portable. This decoupling also means that developers can implement their own operators and use them with other Nektar++ components, without having to make changes to Nektar itself.

All tests added to the code as part of the merge request (for both CPU and CUDA-based operators) have been confirmed to execute successfully by a UKAEA reviewer.

While the grantee only committed to adding (refactored) CPU-based implementations for each operator, CUDA versions, suitable for execution on NVIDIA GPUs, have also been written in several cases. The overall effect of these changes is to make the software more suitable for developing Exascale-capable applications.

6 Summary

This report has presented results from a number of workstreams related to the implementation of 3-D plasma turbulence equations in a realistic tokamak geometry using finite element techniques.

A variety of test cases were implemented using the Firedrake finite element code, demonstrating its utility as a tool for rapid prototyping as well as for experimenting with different numerical implementations and debugging existing NEPTUNE Proxyapps.

An implementation of the 3-D Hasegawa-Wakatani equations using Nektar++ was demonstrated, intended as a stepping stone to a more complex set of fluid plasma equations. The code extends existing Nektar++ solvers embedded in NESO whilst retaining existing functionality (in particular, coupling to a kinetic neutrals framework) via a substantial code refactoring exercise.

The report also described NESO-fame, a tool for generating field-aligned meshes. Particular focuses of this work were on producing meshes that can conform to the first wall of the tokamak and that are of sufficient quality to use in scalable fluid plasma simulations.

Finally, summaries were provided for a number of grantee deliverables, covering ongoing improvements to the Oxford grantee's finite element drift-kinetic model and various updates to the Nektar++ codebase that enhance its suitability for modelling fusion-relevant problems on Exascale platforms.

Acknowledgement

The support of the UK Meteorological Office and Strategic Priorities Fund is acknowledged. ET acknowledges assistance from the rest of the UKAEA NEPTUNE team, as well as Hussam al-Daas (STFC), Professor Patrick Farrell (Oxford) and Professors Colin Cotter and David Ham of Imperial College London.

References

[1] Florian Rathgeber, David A. Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew T. T. Mcrae, Gheorghe-Teodor Bercea, Graham R. Markall, and Paul H. J. Kelly. *Firedrake*: automating the finite element method by composing abstractions. *ACM Trans. Math. Softw., 43(3):24:124:27, 2016. URL: http://arxiv.org/abs/1501.01809, arXiv:1501.01809, doi:10.1145/2998441.*, 2015.

[2] Patrick E. Farrell, Robert C. Kirby, and Jorge Marchena-Menendez. Irksome: Automating runge–kutta time-stepping for finite element methods, 2020.

[3] DG advection with upwinding. `https://www.firedrakeproject.org/demos/DG_advection.py.html`. Accessed: March 2024.

[4] Busch K., Koenig M., and Niegemann J. Discontinuous Galerkin methods in nanophotonics. *Laser and Photonics Reviews Vol.5 Issue 6 (Nov 2011).*, pages 773–809, 2011.

[5] Nektar-Driftwave. `https://github.com/ExCALIBUR-NEPTUNE/nektar-driftwave`. Accessed: March 2023.

[6] E. Threlfall and W. Arter. Finite Element Models Complementary Actions: Code Integration, Integration and Operation 3. Technical Report CD/EXCALIBUR-FMS/0079, UKAEA Project Neptune, 2023.

[7] m6c5 scripts. `https://github.com/ethrelfall/m6c5_phase2/tree/main`, 2024. Accessed: March 2024.

[8] Hermes-3. `https://hermes3.readthedocs.io/en/latest/examples.html#d-drift-plane`. Accessed: March 2023.

[9] STORM. `https://github.com/boutproject/STORM`, 2024. Accessed: March 2024.

[10] W. Arter. Equations for ExCALIBUR / NEPTUNE proxyapps. Technical Report CD/EXCALIBUR-FMS/0021, UKAEA Project Neptune, 2023. `https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/ukaea_reports/CD-EXCALIBUR-FMS0021-1.26-M1.2.1.pdf`.

[11] Donea J. and Huerta A. *Finite Element Methods for Flow Problems*. Wiley, 2003.

[12] Bernsen E., Bokhove O., and van der Vegt J.J.W. A (Dis)continuous finite element model for generalized 2D vorticity dynamics. *Memorandum No. 1787, Department of Applied Mathematics, University of Twente*, 2005.

[13] Nektar-Driftplane. `https://github.com/ExCALIBUR-NEPTUNE/nektar-driftplane`. Accessed: March 2023.

[14] E. Threlfall, J. Cook, M. Barton, O. Parry, W. Saunders, and W. Arter. Complementary actions. Code integration, acceptance and operation 3. Technical Report CD/EXCALIBUR-FMS/0074-M6c.3, UKAEA, 3 2023. `https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/ukaea_reports/CD-EXCALIBUR-FMS0074-M6c.3.pdf`.

[15] E. Threlfall and W. Saunders. Support high-dimensional procurement. Technical Report CD/EXCALIBUR-FMS/0066, UKAEA Project Neptune, 2022. https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/ukaea_reports/CD-EXCALIBUR-FMS0066-M4.1.pdf.

[16] E. Threlfall, O. Parry, and W. Arter. Finite Element Models Complementary Actions: Code Integration, Integration and Operation 3. Technical Report CD/EXCALIBUR-FMS/0074, UKAEA Project Neptune, 2023.

[17] J. Cook, W. Saunders, and O. Parry. Three-Dimensional integrated particle and continuum model. Technical Report CD/EXCALIBUR-FMS/0079-M4c.3, UKAEA, 9 2023. https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/ukaea_reports/CD-EXCALIBUR-FMS0079-M4c.3.pdf to be posted.

[18] B.D. Dudson. BOUT++ repository. https://github.com/boutproject/BOUT-dev, 2020. Accessed: June 2020.

[19] R. Numata, R. Ball, and R.L. Dewar. Bifurcation in electrostatic resistive drift wave turbulence. *Physics of Plasmas*, 14:102312, 2007.

[20] B. Friedman and T. A. Carter. A non-modal analytical method to predict turbulent properties applied to the Hasegawa-Wakatani model. *Physics of Plasmas*, 22(1):012307, 01 2015.

[21] W. Arter et al. Equations for EXCALIBUR/NEPTUNE Proxyapps. Technical Report CD/EXCALIBUR-FMS/0021-1.30-M1.2.1, UKAEA, 6 2023. https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/ukaea_reports/CD-EXCALIBUR-FMS0021-1.30-M1.2.1.pdf.

[22] F.F. Chen. *Introduction to Plasma Physics, 3rd ed.* Springer Science & Business Media, 2012.

[23] A. Roques and open source developers. Plantuml website. https://plantuml.com/, 2024. Accessed: February 2024.

[24] T. Marin. hpp2plantuml python package. https://pypi.org/project/hpp2plantuml/, 2024. Accessed: February 2024.

[25] D. Napoleone, S. Marks, and E. Frederich. plantuml python package. https://pypi.org/project/plantuml/, 2024. Accessed: February 2024.

[26] J. Cook, E. Threlfall, , O. Parry, W. Saunders, and W. Arter. Three-Dimensional integrated particle and continuum model continued. Technical Report CD/EXCALIBUR-FMS/0082-M4c.4, UKAEA, 3 2024. https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/ukaea_reports/CD-EXCALIBUR-FMS0082-M4c.4.pdf.

[27] G. Karniadakis and S. Sherwin. *Spectral/hp element methods for computational fluid dynamics 2nd Ed.* Oxford University Press, 2005. https://doi.org/10.1093/acprof:oso/9780198528692.001.0001.

[28] J. W. Cook, J. T. Parker, W. R. Saunders, and Parry O. Neptune Exploratory SOftware (NESO) repository. https://github.com/ExCALIBUR-NEPTUNE/NESO, 2023. Accessed: January 2023.

[29] C. Ridgers and M. Kryjak. Benchmarks for basic turbulence cases (Report on Test Cases and Proxy-App) - updated . Technical Report 2067270-TN-06, UKAEA Project Neptune, 2024. to be posted at https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2067270/TN-06.pdf.

[30] Edward Laughton, Gavin Tabor, and David Moxey. A comparison of interpolation techniques for non-conformal high-order discontinuous galerkin methods. *Computer Methods in Applied Mechanics and Engineering*, 381:113820, 2021.

[31] John Omotani, Ben Dudson, Peter Hill, Vandoo, and bshanahan. boutproject/hypnotoad: 0.5.2, March 2023.

[32] Nektar-diffusion proxyapp. https://github.com/ExCALIBUR-NEPTUNE/nektar-diffusion, 2021. Accessed: September 2021.

[33] M. Barnes, M.R. Hardman, S.L. Newton, J. Omotani, and F.I. Parra. A reduced electron model for testing plasma dynamics for closed field lines. Technical Report 2070839-TN-03, UKAEA Project Neptune, 2023. https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2070839/TN-03.pdf.

[34] M.R. Hardman, J. Omotani, M. Barnes, S.L. Newton, and F.I. Parra. Convergence of the solutions of the 2D-1V ion drift kinetic equation with wall boundary conditions: manufactured solutions tests. Technical Report 2070839-TN-04, UKAEA Project Neptune, 2023. https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2070839/TN-04.pdf.

[35] G. Strang and G.J. Fix. *An Analysis of the Finite Element Method*. Prentice-Hall, Englewood Cliffs, NJ, 1973.

[36] J.P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover Publications, Mineola, NY, 2001. http://www-personal.umich.edu/ jpboyd/BOOK_Spectral2000.html.

[37] M. Barnes, M.R. Hardman, S.L. Newton, J. Omotani, and F.I. Parra. Numerical implementation of a fluid model for electrons in a drift kinetic code. Technical Report 2070839-TN-05, UKAEA Project Neptune, 2023. https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2070839/TN-05.pdf.

[38] M.R. Hardman, J. Omotani, M. Barnes, S.L. Newton, and F.I. Parra. Ion-ion model collision operators: a Krook operator and a model Fokker-Planck operator. Technical Report 2070839-TN-06, UKAEA Project Neptune, 2023. https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2070839/TN-06.pdf.

[39] M.R. Hardman, M. Abazorius, J. Omotani, M. Barnes, S.L. Newton, and F.I. Parra. A higher-order finite-element implementation of the full-F Landau Fokker-Planck collision operator for charged particle collisions in a low density plasma. Technical Report 2070839-TN-07, UKAEA Project Neptune, 2023. https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2070839/TN-07.pdf.

[40] M. Barnes, M.R. Hardman, S.L. Newton, J. Omotani, and F.I. Parra. Numerical implementation of moment-kinetic electrons. Technical Report 2070839-TN-08, UKAEA Project Neptune, 2023. to be posted at https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2070839/TN-08.pdf.

[41] M.R. Hardman, J. Omotani, M. Barnes, S.L. Newton, and F.I. Parra. Aspects of magnetic geometry in a drift-kinetic code with wall boundaries. Technical Report 2070839-TN-09, UKAEA Project Neptune, 2024. to be posted at https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2070839/TN-09.pdf.

[42] M. Barnes, J. Omotani, S.L. Newton, M.R. Hardman, and F.I. Parra. An update on numerical implementation of moment-kinetic electrons. Technical Report 2070839-TN-10, UKAEA Project Neptune, 2024. to be posted at https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2070839/TN-10.pdf.

[43] M. Barnes, M.R. Hardman, S.L. Newton, J. Omotani, and F.I. Parra. Numerical implementation of moment-kinetic electrons. Technical Report 2070839-TN-08, UKAEA Project Neptune, 2024. to be posted at https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2070839/TN-08.pdf.

[44] D. Moxey, M. Green, C. Cantwell, and S. Sherwin. DG implementation for Blob2D equations. Technical Report 2068435-TN-01, UKAEA Project Neptune, 2024. to be posted at https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2068435/TN-01.pdf.

[45] D. Moxey, M. Green, C. Cantwell, K. Kirilov, J. Zhou, J. Peiro, and S. Sherwin. Summary of mesh generation activities. Technical Report 2068435-TN-02, UKAEA Project Neptune, 2024. to be posted at https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/2068435/TN-02.pdf.