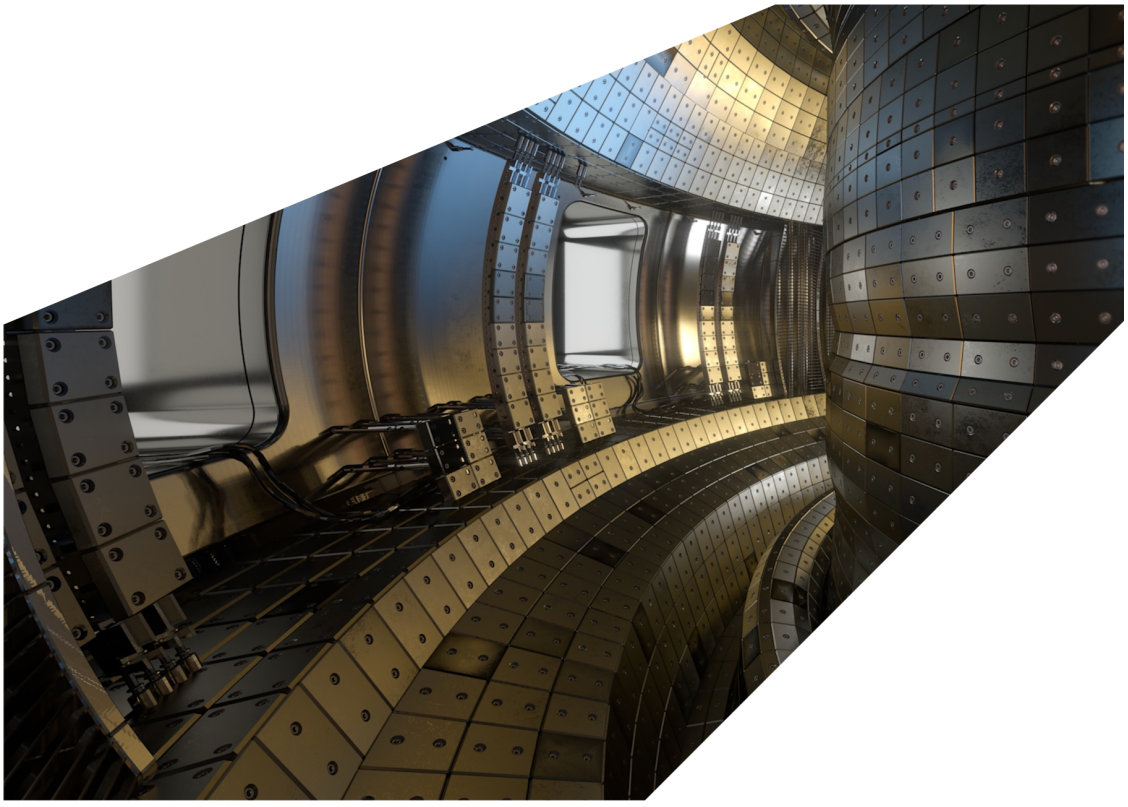
UK Atomic Energy Authority

ExCALIBUR

Finite Element Models: Performance

M2.2.2

Abstract
This report describes work for ExCALIBUR project NEPTUNE at Milestone 2.2.2. It includes preliminary evaluations of the *Nektar++*-based NEPTUNE proxyapps *Nektar-Diffusion* (intended to simulate diffusion problems involving anisotropic and spatially-varying diffusivity tensors) and *Nektar-Driftwave* (intended to solve the Hasegawa-Wakatani equations); these assessments are intended as quality checks, rather than guarantees of accuracy. An additional section contains a discussion of *Soldrake*, a one-dimensional continuum model of the scrape-off layer implemented in the *Firedrake* software and using the *Irksome* library for time-evolution.



UKAEA REFERENCE AND APPROVAL SHEET

	Client Reference:	
	UKAEA Reference:	CD/EXCALIBUR-FMS/0047
	Issue:	1.00
	Date:	September 29, 2021
Project Name: ExCALIBUR Fusion Modelling System		

	Name and Department	Signature	Date
Prepared By:	Will Saunders Ed Threlfall BD	N/A N/A	September 29, 2021 September 29, 2021
Reviewed By:	Rob Akers Advanced Computing Dept. Manager		September 29, 2021

1 Introduction

The NEPTUNE project aims to leverage high-order techniques in challenging plasma edge applications, ultimately targetting forthcoming exascale hardware. These methods offer, in ideal cases, exponential convergence for polynomial increases in compute time, and are architecturally suited to modern heterogeneous computing methods. This report concerns high-order finite element implementations of some of the NEPTUNE equations, specifically, those representing continuum equilibrium (i.e. non-kinetic) fluids.

The modern open-source C++ spectral/hp element framework *Nektar++* offers a flexible and powerful framework for the development of new solvers [1]. *Nektar++* is thus the basis for two of the NEPTUNE proxyapps: *Nektar-Diffusion* [2] and *Nektar-Driftwave* [3], which simulate respectively diffusion with a spatially-varying anisotropic diffusivity tensor and the Hasegawa-Wakatani system for modelling turbulent drift-wave dynamics. In this report, these proxyapps are subjected to some initial testing and a brief examination of coding standards with the aim of verifying a minimal quality standard.

Another open-source finite-element framework is *Firedrake* [4], which provides also a UFL-based domain-specific language that allows the user the facility to implement equations (which must be expressed in weak form) that are then solved by the underlying codebase using automatic code generation of C code and the potential for multiple backends. This software forms the basis for the *Soldrake* implementation of a one-dimensional continuum model of the scrape-off layer. These investigations use also the complementary time-stepping extension *Irksome* [5] to examine the dynamics of electron and ion density, velocity, and temperature along a magnetic field line bounded on both ends by a divertor.

2 Performance of *Nektar-Diffusion* proxyapp

2.1 Theory

The theory of classical transport in a magnetized plasma leads to two transport coefficients governing heat transfer in the directions parallel with and perpendicular to the applied magnetic field [6]. The quantities

$$\kappa_{\parallel} = 19.2\sqrt{2\pi^3}\frac{1}{\sqrt{m_e}}\frac{\epsilon_0^2}{e^4}\frac{(k_B T_e)^{5/2}}{Z^2\lambda} \quad \kappa_{\perp} = \frac{1}{6\sqrt{\pi^3}}\frac{1}{m_i}\Big(\frac{nZe}{B\epsilon_0}\Big)^2\frac{(m_p A)^{3/2}\lambda}{\sqrt{k_B T_i}} \tag{1}$$

are converted into diffusivities by multiplying by a factor of $\frac{2}{3n_e}$ where n_e is the density of electrons. Then the equation to be solved by the proxyapp is

$$\frac{\partial u}{\partial t} = \nabla \cdot (D\ \nabla u)\,. \tag{2}$$

The components of the diffusivity tensor D are rotated to allow arbitrary alignment between the magnetic axis and the coordinate frame used by the code (and the boundaries). This tensor is a second-rank contravariant object (note ∇ is covariant; note also that in the Cartesian frames used the covariant / contravariant distinction does not matter anyway) and so the transformation law between frames of reference (which are, in practice, rotations) is, with the usual Einstein summation convention,

$$D^{i'j'} = \frac{\partial x^{i'}}{\partial x^i}\frac{\partial x^{j'}}{\partial x^j}D^{ij}. \tag{3}$$

Note that in three dimensions, there are two independent directions perpendicular to the magnetic axis and in this subspace the components of D take the form of a general two-dimensional isotropic tensor viz. a linear combination of $g^{\perp}_{ab}$ and $\epsilon^{\perp}_{ab}$, respectively the metric and alternating tensor in the two-dimensional perpendicular space.

2.2 Code structure and style

The proxyapp takes the $\kappa_{\parallel}$ and $\kappa_{\perp}$ as inputs via the *Nektar++* `xml` session file, as well as an angle to specify the magnetic axis (whose use is illustrated in 2.3). The κ coefficients are, in the examples provided, calculated, using the Braginskii formulae 1, from physical constants and parameters provided in the session file. These are converted to diffusivities, as shown above, and used as inputs to an existing part of the *Nektar++* code.

There are three main components (C++ translation units) to the proxyapp, handling the steady-state, time-varying, and spatially-varying magnetic fields cases (additionally, the spatially-varying diffusivity tensor has been added to the main *Nektar++* code base as part of this work). The style of the new code is consistent with the broader *Nektar++* style i.e. good-standard modern C++.

There is not a great deal of novel code in the proxyapp, since the main work is done by the underlying *Nektar++* code base. This has the benefit of using established, tested code. The modular nature of the framework means that it is straightforward to alter other aspects of the problem, for example changing the computational mesh in order to study the numerical effects of misalignment between the mesh and the magnetic axis. This dependency also means that the HPC viability and performance (also performance portability) of the proxyapp are determined by that of *Nektar++* and will benefit from other work in this direction e.g. the forthcoming matrix-free accelerated kernels and the preconditioning aspect of NEPTUNE. In the meantime, existing examinations of the performance of the framework are expected to be relevant [7].

Three example cases are provided by the authors of the proxyapp; these are studied below. One point is that all examples have $\kappa_\perp$ strictly zero (actually not a bad first approximation to the transport properties of a well-magnetized plasma, but it does mean that there is functionality that is not tested by the examples). This omission is addressed by further testing in this report (see 2.7, 2.8).

2.3 Examples I: steady-state

The steady-state solver finds solutions to (potentially anisotropic) Laplace equations.

In this example, a steady state is found; the field is sourced on the boundary, the left-hand-side having a Dirichlet condition corresponding to a peaked one-dimensional density profile, with all other boundaries using a zero Neumann condition. The value of $\kappa_\perp$ is set to zero. The example uses a mesh of 25,600 2D squares with global element order $p = 3$ (which controls the number of intra-element degrees of freedom) and it executes in c.20s on a single modern desktop PC.

Examples with tilt angles of zero and two degrees are included (the two-degree is shown in fig.1). Rotating the angle has the effect of changing the magnetic axis: a positive rotation angle has the effect of aiming the beam in a downward tilt. Since the domains are currently featureless, any effects of varying the magnetic axis are due to interaction with the boundary conditions. This means that the zero-degree case is near-analytic (the boundary profile is simply extruded along the magnetic axis, subject to minor corrections from the Neumann constraints at the top and bottom walls) whereas the inclined case is not, due to its non-normal boundary incidence.

It may be noted that the zero-Neumann boundary condition used where the diffusing field encounters the boundary does not correspond to an outflow boundary condition, i.e. there will be some reflection of the field at this location (see fig.2).

It is noted that the overall scalar magnitude governing the diffusion tensor does not affect the steady state, so the magnitude of $\kappa_\parallel$ is irrelevant in this example.

2.4 Examples II: time-evolving state

This example begins with an empty cavity with the same boundary conditions as used in the previous example; the Dirichlet source term on the left-hand-side boundary means that the field diffuses into the cavity with time, eventually tending to the steady state found in 2.3.

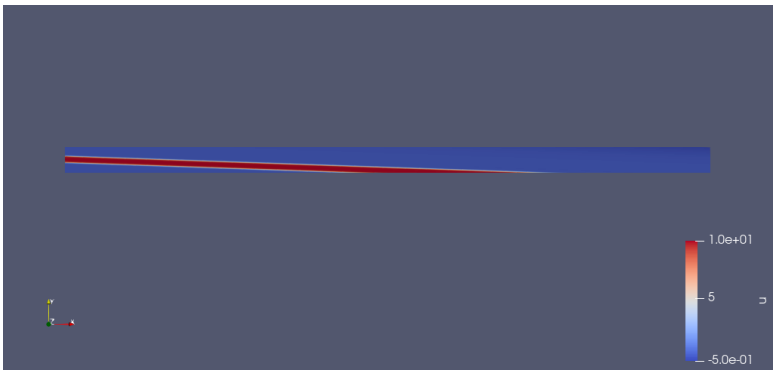


Figure 1: Output of the anisotropic solver ($\kappa_\perp = 0$) for the steady state, in the case of two degrees of magnetic axis tilt.

The parallel diffusivity evaluates as $3.140868 \times 10^{-6} m^2 s^{-1}$ using $n_e = 1.0 \times 10^{18} m^{-3}$. This value was exported from the code for checking and was found to agree with eq.1. The magnitude here governs the speed of the time-evolution. One issue with the example is that it is set up to run for one program time unit (corresponding to one second, as SI units are used), using a time step Δt of 1ms. These timescales are much longer than the dynamical timescale implied by the lengthscale and diffusion coefficient in the problem (this timescale is $\frac{L^2}{D}$ for lengthscale L and a scalar diffusivity D, which is calculated as $\frac{2\kappa_\parallel}{3n_e}$) and this has the effect of making the problem appear steady-state, as this state is achieved after a single time-step. More sensible are $\Delta t = 10^{-9} s$ and a duration of a few times $10^{-5} s$, over which significant build-up of u at the right-most boundary is achieved.

The zero-degree example here constitutes a one-dimensional problem; conceptually the Dirichlet condition acts as a steady source, with the value of the field held constant along the left-hand-side boundary, with a Neumann condition at the right-hand-side boundary representing a zero-flux condition (meaning that the steady state is just a uniform field). In fact, a similar time-dependent boundary value problem ($\dot{u} = Du''$) has a simple analytic solution on the semi-infinite domain $x \geq 0$, viz.

$$u = 10 \left(1 - \text{erf} \left(\frac{x}{\sqrt{4Dt}} \right) \right). \tag{4}$$

Here the amplitude of 10 is chosen to correspond to the field value in the example. The field along the centre of the test profile of the example can be plotted at various times and compared to this (the approximation is good while the field value at the right-hand-side boundary is small i.e. $t \lesssim \frac{L^2}{16D}$ for a domain of length L). The correspondence (fig.2) is obvious; note the $t = 10^{-6} s$ case shows a small amount of reflection due to the no-flux boundary condition on the right-hand side.

2.5 Examples III: torus

This example demonstrates the spatial variation of the anisotropic tensor, used here to represent alignment with a semicircular magnetic axis. The mesh is quadrilateral with order-three elements.

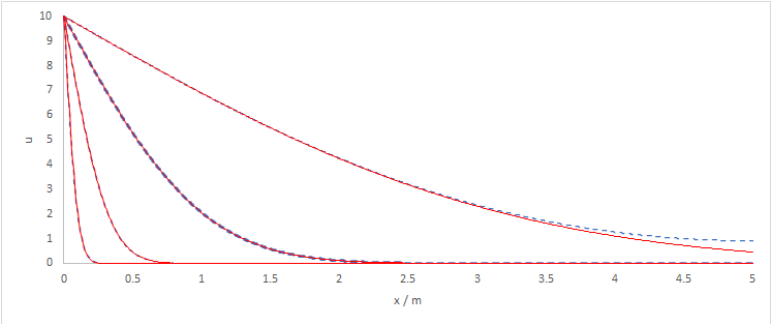


Figure 2: Time evolution: output of the anisotropic solver (dashed curves) overlain with theory (eq.4) for times 10^{-9},10^{-8}, 10^{-7}, $10^{-6}s$ respectively from left to right. The latest-time numerics deviate from the semi-infinite theory due to the presence of the right-hand-side boundary.

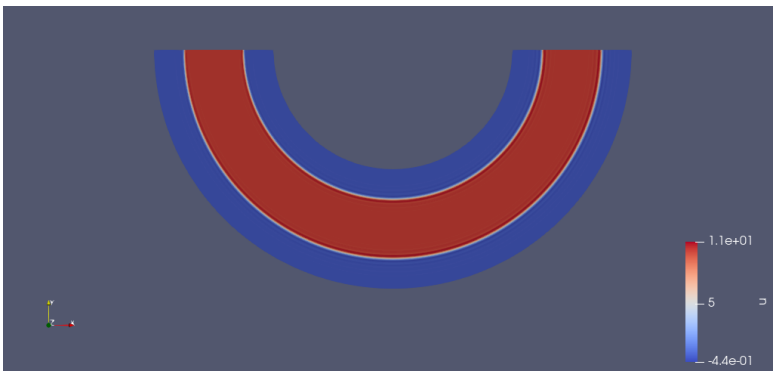


Figure 3: Output of the anisotropic solver ($\kappa_\perp = 0$) for the steady state of a toroidally-aligned magnetic axis.

The output demonstrates that the spatially-varying code is working in this two-dimensional example. The generated field is shown in fig.3.

2.6 Further tests I: spectral convergence

One of the outputs of a *Nektar++* simulation is an assessment of the global L2 error against a reference analytic solution. It is therefore possible to test the convergence of a known stationary analytic solution, regarding which it is trivial to generate solutions of the 2D Laplace equation by taking the real or imaginary parts of analytic functions. Here we examine the convergence of the isotropic case (no magnetic axis) in the case of the function $\cos(2x)\cosh(2y)$ (the real part of $f(z) = \cos(2z)$). Dirichlet boundary conditions are chosen consistent with the solution. The system is solved for element orders $p = 1 - 11$ using an isotropic diffusion tensor. A plot of the behaviour of the logarithm of the global L2 error against element order (fig.4) shows a clear pattern of exponential convergence, which saturates in this case for $p = 9$. Note that for this simulation, and the ones in the following subsections, the computational domain was the unit square centred on the origin, discretized into 100 identical squares.

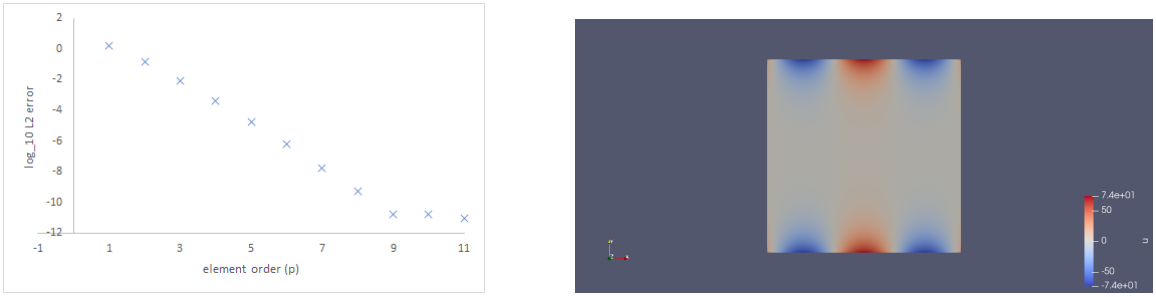


Figure 4: Plot of global L2 error from simulation of harmonic function as a function of element order, showing exponential convergence (left). On the right is the output of the solver for element order $p = 11$.

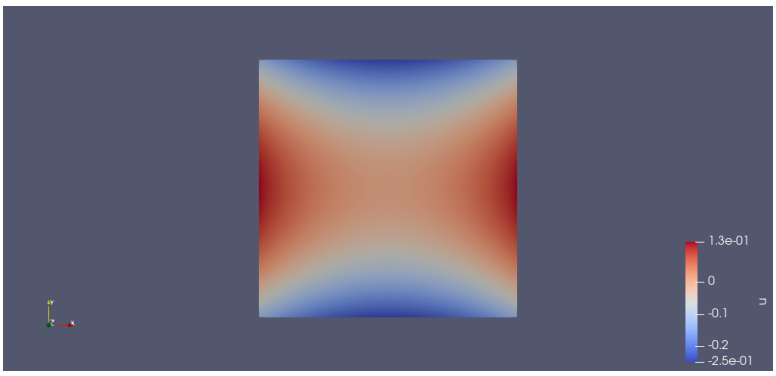


Figure 5: Output of the anisotropic solver showing the solution as a stretched harmonic function.

2.7 Further tests II: anisotropic field

The following tests (this, and the following, subsections) used element order $p = 3$.

It is possible to test the validity of the anisotropic solver by setting up test cases that represent solutions to Laplace's equation in 2D in coordinates that are stretched. It is further possible to assess the error in the solution by direct comparison with the analytic solution, as used in the preceding section.

Selecting as a simple smooth trial solution for u the harmonic function $x^2 - y^2$ (the real part of $f(z) = z^2$), one can see that the stretched function $\left(\frac{x}{a}\right)^2 - y^2$ is a solution to the equation $a^2 \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$. A simulation with $\kappa_{\parallel} = 2.0$ and $\kappa_{\perp} = 1.0$ was performed and the output is shown in fig.5.

The global L2 error remains of the order of 10^{-11} for the anisotropic solution, consistent with the 'saturation' seen earlier (2.6).

A further note on the performance is that the field in 2.6 was run using the method in this section for values of $\frac{\kappa_{\parallel}}{\kappa_{\perp}}$ ranging from 10^{-10} to 10^{10} apparently without any effect on the efficiency or accuracy of the code.

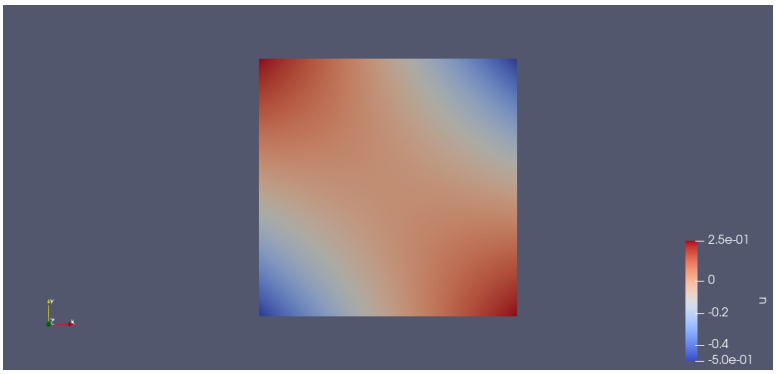


Figure 6: Output of the anisotropic solver showing the solution as a stretched, rotated harmonic function.

2.8 Further tests III: rotated anisotropic field

It is possible to test the rotation of the magnetic axis in a simple analytic case. A 45-degree tilt can be applied to the magnetic axis and the Dirichlet boundary conditions and the reference solution can be made consistent with a one-dimensional steady-state diffusive solution $u = x^2 - y^2$ in rotated coordinates. It is found that the required solution is obtained, showing correct transformation of the diffusion tensor in the code; as before, the global L2 error remains consistent with the saturation level and appears agnostic as to the choice of rotation angle. The field is shown in fig.6, which is clearly a rotated view of the field in fig.5.

Amusingly, for the case of linear functions, one notes that the equation system is insensitive to the rotation angle, because any linear function satisfies $\frac{\partial^2 u}{\partial x^2} = 0$ and $\frac{\partial^2 u}{\partial y^2} = 0$ independently. The solution in this case is fully specified by the boundary conditions alone (i.e. the magnetic axis does not need to coincide with the gradient of u). One issue here is that if the *Nektar++* variable `GlobalSysSoln` is set to `DirectFull`, an incorrect result is obtained for the linear function case. A correct solution follows if this option is set to `IterativeStaticCond`.

3 Behaviour of *Nektar-Driftwave* proxyapp

This proxyapp, again based on the *Nektar++* framework, is designed to solve the 2D Hasegawa-Wakatani equations [8] - a coupled pair of equations for the plasma density and the electrostatic potential found in resistive drift-wave turbulence, used to model plasma edge turbulence. The equations are (potential ϕ, density n, vorticity ζ; square bracket is the Poisson bracket)

$$\frac{\partial \zeta}{\partial t} + [\phi, \zeta] = \alpha (\phi - n), \tag{5}$$

$$\frac{\partial n}{\partial t} + [\phi, n] = \alpha (\phi - n) - \kappa_n \frac{\partial \phi}{\partial y}. \tag{6}$$

Here, κ_n is a constant-density gradient scale-length and α is a parameter called the adiabaticity.

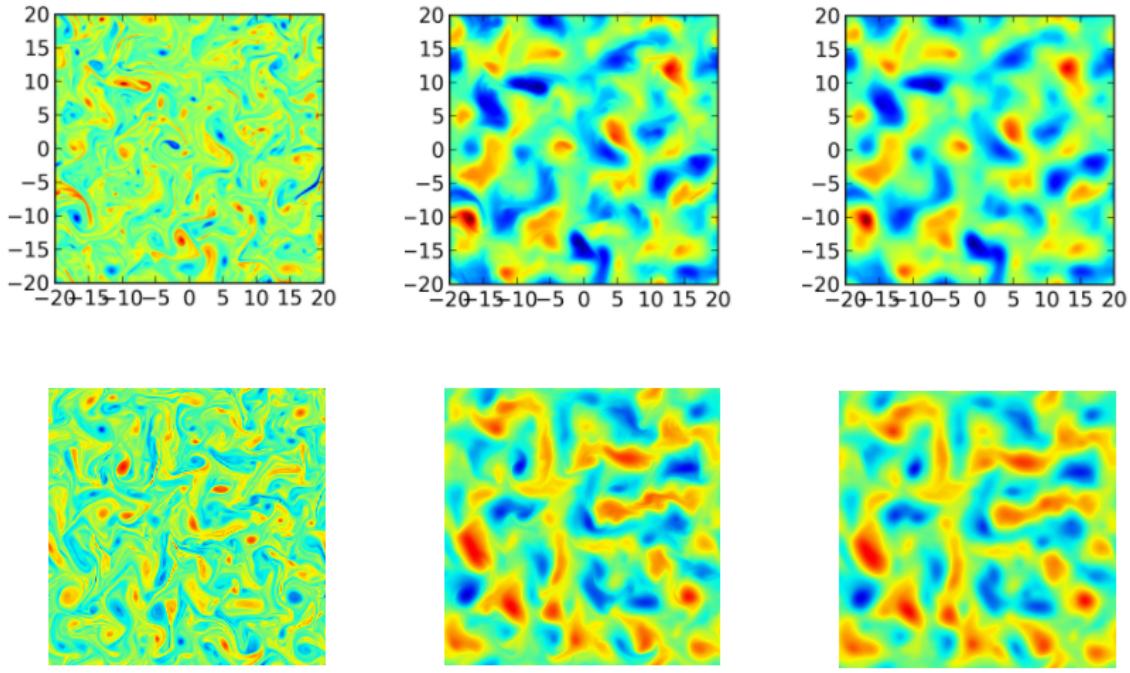


Figure 7: Example simulated vorticity, density, and electrostatic potential (left to right) from Ammar Hakim's web page (ref) (top) with corresponding proxyapp outputs (bottom), showing qualitative correspondence.

The electrostatic potential ϕ is solved by the Poisson system $\nabla^2\phi = \zeta$ (note that *Nektar++* has inbuilt capability to handle this type of equation as it is found in many fluid flows e.g. for the pressure in the incompressible Navier-Stokes system).

The provided example solves the time-evolution of this system starting from Gaussian initial profiles, using a 64×64 grid with element order $p = 3$. The output includes check-point fields at regular intervals during the solution history. One issue is that the example is set to run for 50 time units, whereas the system is still clearly in a transient state at the end of this period. Increasing the duration to 200 time units allows comparison with the adiabaticity-2.0 results from the webpage [9]. These results, together with the corresponding outputs of the proxyapp, are shown in fig.7. The qualitative agreement is impressive; note that predicting the specific structures is not possible due to the chaotic nature of the system - thus, a L2-error assessment against the field of a reference solution is not applicable - one must instead examine the existence and statistics of structures such as vortices (evident in both cases). One further thing to note is that the electrostatic potential is seen to correlate well with the density profile. It must be stressed that this test forms a baseline check of the proxyapp and does not provide a definitive guarantee of accuracy.

This longer simulation executed in approximately eight hours on a single modern desktop PC using eight cores (it is assumed that the specified time-step size cannot be increased in order to reduce the computation time). HPC performance is, as with *Nektar-Diffusion*, determined by the underlying *Nektar++* framework.

The coding style of the proxyapp is consistent with that of *Nektar++*.

4 Soldrake

4.1 Introduction

Transport in the scrape-off layer (SOL) occurs primarily along magnetic field lines. The field lines considered here are taken to be outside the last closed flux surface; they therefore terminate in a divertor at both ends. For modelling transport along a field line, the latter is considered to be a symmetric flux tube where plasma enters the tube in the perpendicular direction and is then transported along the tube to the divertor which acts as the only sink for plasma. At the ends of the tube, where the flux line meets the divertor, a scenario where plasma is extracted at a prescribed velocity in considered. This study uses an established model for the plasma as a fluid without neutral species; the model describes the evolution of a plasma described by density, velocity and temperature with boundary conditions and sources that determine the profile of the fields. Of particular interest are combinations of source terms and boundary conditions that produce high spatial gradients in the steady state solution as these scenarios are the most challenging for a numerical implementation to capture accurately.

Typically in the plasma physics community the method of finite differences has been used as a spatial discretization technique [10, 11, 12]. Here, the finite element method is used as the spatial discretization method, as this is an area marked for investigation by the NEPTUNE project - in particular, the investigation of simulation accuracy in response to the choice of spatial discretization, where refinement can occur both through mesh refinement (h-adaptivity) and polynomial order refinement (p-adaptivity). Both refinement types increase the number of degrees of freedom (DOFs) used to represent the solution with the expectation that applying more DOFs to a computation results in a more accurate solution. An important target for NEPTUNE is to establish how many DOFs are required to compute a solution to a given error tolerance.

As finite element implementations are highly non-trivial to implement, the *Firedrake* [4] framework was used to easily implement our model equations from weak form. For the time discretization approach, the *Irksome* [5] extension to the *Firedrake* framework was used; this provides an easy method to implement general Runge-Kutta schemes. In the following sections the model equations and initial results for systems without neutral species are described. These results demonstrate that the numerical implementation is correct, and allow investigation of simulation output accuracy as a function of spatial discretization.

4.1.1 *Firedrake* and *Irksome*

Firedrake is a framework that automates the computation of finite element based solutions of partial differential equations (PDEs). Users describe a system of equations along with a domain and boundary conditions in a high-level language called Unified Form Language (UFL). The *Firedrake* framework consumes the UFL input and automatically uses code generation techniques to efficiently conduct lower level operations such as matrix assembly. Linear and non-linear systems are efficiently and automatically solved by the framework by using the PETSc[13] library.

For the canonical example, consider a 1D Poisson equation on the interval domain $\Omega = [0, 1]$

where

$$-\Delta U = f \text{ in } \Omega, \tag{7}$$
$$U = 0 \text{ on } \partial\Omega. \tag{8}$$

The corresponding weak formulation is to find a $u \in V$ such that

$$\int_\Omega \nabla\phi \cdot \nabla u \, dx = \int_\Omega \phi f \, dx \quad \forall \, \phi \in V \tag{9}$$

where V is a function space that satisfies the prescribed boundary condition. The Python code in Listing 1 demonstrates how this weak form would be described by UFL. Note that with this approach the function space and weak forms themselves can be easily modified for experimentation and development. This is in contrast to a traditional monolithic code where alterations to function space or functional form may easily result in a significant implementation cost to realize.

Irksome is a complementary project in the *Firedrake* ecosystem which provides high-level access to Runge-Kutta integration schemes. In this work, implicit schemes that offer high stability properties are of particular interest. To allow users to implement time stepping at a high level, the *Irksome* library implements an extension to UFL that provides the `Dt` operator which the user includes in the time-dependent weak form of interest.

Listing 1: Implementation in UFL of the weak formulation of Poisson's equation on a unit interval.

```
from firedrake import *
mesh = UnitIntervalMesh(100)
V = FunctionSpace(mesh, "CG", 2)
u = TrialFunction(V)
v = TestFunction(V)
a = inner(grad(u), grad(v)) * dx
F = Function(V)
F.interpolate(<construct F>)
L = F*v*dx
bcs = [DirichletBC(V, Constant(0), (1,)),
     DirichletBC(V, Constant(0), (2,)), ]
uu = Function(V)
solve(a == L, uu, bcs=bcs)
```

4.2 1D scrape-off layer model

4.2.1 Model equations

Consider a 1D domain of length L parameterized by the variable $s \in [0, L]$ and parameterize time by t. The equations describing conservation of mass, momentum, and energy are

$$U_r \frac{\partial n}{\partial t} = -\frac{\partial}{\partial s}(nu) + S^n, \tag{10}$$

$$U_r \frac{\partial}{\partial t}(nu) = -\frac{\partial}{\partial s}(nu^2) - \frac{\partial}{\partial s}(nT) + S^u, \tag{11}$$

$$U_r \frac{\partial}{\partial t}\left((g-2)nT + nu^2\right) = -\frac{\partial}{\partial s}(gnuT + nu^3) + \kappa_d \frac{\partial^2 T}{\partial s^2} + S^E, \tag{12}$$

where $n = n(s,t)$ is the ion and electron density, $u = u(s,t)$ is the velocity field of ions and electrons, and $T = T(s,t)$ is the temperature of ions and electrons. The constant g represents specific heat capacity. The temperature diffusion coefficient κ_d typically takes the form $\kappa_d = \kappa_0 T^{5/2}$ or can be set to zero in the conduction-free scenario. The constant U_r, which measures the importance of the transient term, will be set to $U_r = 1$. The terms $S^n(s)$, $S^u(s)$ and $S^E(s)$ are time-independent sources which represent the addition or removal of quantities in the direction perpendicular to the flux tube. For boundary conditions it is assumed that either $\frac{\partial *}{\partial s} = 0$ for $* \in \{n, u, T\}$ on $s \in \{0, 1\}$ or that the exit velocity $u(0) = -u(1)$ is fixed at a constant value. In the latter case, where the velocity is fixed on the boundary, it follows from applying conservation of mass and energy that by choosing an exit velocity the boundary values of mass and temperature are also determined. Note that these equations are written in the non-dimensionalized form described by Arter [14]. For a given set of source terms, quantities are normalized such that Dirichlet boundary conditions could be applied where n, u and T all have magnitude 1 at the boundary.

By applying the product rule repeatedly and replacing time derivatives with spatial derivatives and forcing terms Equations 11 and 12 are rewritten in the so-called non-conservative form,

$$U_r n \frac{\partial u}{\partial t} = u\left[\frac{\partial}{\partial s}(nu) - S^n\right] - \frac{\partial}{\partial s}(nu^2) - \frac{\partial}{\partial s}(nT) + S^u, \tag{13}$$

$$= -uS^n - nu\frac{\partial u}{\partial s} - \frac{\partial}{\partial s}(nT) + S^u \tag{14}$$

and

$$\begin{aligned} U_r(g-2)n\frac{\partial T}{\partial t} = \left((g-2)T - u^2\right)\left[\frac{\partial}{\partial s}(nu) - S^n\right] + 2u\left[\frac{\partial}{\partial s}(nu^2) + \frac{\partial}{\partial s}(nT) - S^u\right] \\ - \frac{\partial}{\partial s}(gnuT) - \frac{\partial}{\partial s}(nu^3) + \kappa_d \frac{\partial^2 T}{\partial s^2} + S^E \end{aligned} \tag{15}$$

where square brackets have been used to more easily identify the insertion of Equations 10 and 11.

4.2.2 Weak formulation

In order to use the *Firedrake* framework, the system of equations must be written in weak form such that the system can be represented in the Unified Form Language (UFL). The spatial dis-

cretization used here takes the form of a function space V constructed of continuous Lagrange basis functions (Firedrake `CG`) where the polynomial order was varied between 1 and 19 depending on the particular investigation. For a test function ϕ, the weak form of Equations (10), (13) and (15) are simply formed by multiplication by the test function and integrating over the whole domain,

$$U_r\langle\phi, \frac{\partial n}{\partial t}\rangle = \langle\phi, -\frac{\partial}{\partial s}(nu) + S^n\rangle, \tag{16}$$

$$U_r\langle\phi, n\frac{\partial u}{\partial t}\rangle = \langle\phi, u\left[\frac{\partial}{\partial s}(nu) - S^n\right] - \frac{\partial}{\partial s}(nu^2) - \frac{\partial}{\partial s}(nT) + S^u\rangle, \tag{17}$$

$$\begin{aligned} U_r(g-2)\langle\phi, n\frac{\partial T}{\partial t}\rangle &= \langle\phi, \left((g-2)T - u^2\right)\left[\frac{\partial}{\partial s}(nu) - S^n\right]\rangle \\ &+ \langle\phi, 2u\left[\frac{\partial}{\partial s}(nu^2) + \frac{\partial}{\partial s}(nT) - S^u\right]\rangle \\ &+ \langle\phi, -\frac{\partial}{\partial s}(gnuT) - \frac{\partial}{\partial s}(nu^3) + S^E\rangle \\ &- \kappa_d\langle\frac{\partial\phi}{\partial s}, \frac{\partial T}{\partial s}\rangle, \end{aligned} \tag{18}$$

where for brevity the notation $\langle u, v\rangle \int_s uvds$ is employed.

4.2.3 Implementation and results

The computation of steady state solutions is formed as a two stage process. First, a time dependent system is constructed with the desired source terms and an initial condition that satisfies the boundary conditions. This initial condition is evolved using the Implicit Euler time stepping scheme via *Irksome*. It is common for the initial condition and boundary conditions to add spurious contributions to the solution. In order to preserve the accuracy of the steady state solution, these undesirable contributions are suppressed by using a small-in-magnitude diffusion operator every set number of time steps.

After the transients from the initial condition have been either suppressed or transported from the domain the fields relax in a direction that minimizes the residual. These evolved fields are an approximation to the steady state and are used as the initial guess for a non-linear solver that computes the steady state. Given a good initial guess, the non-linear solve computes a solution with an acceptably small residual in a significantly shorter time frame than applying further time stepping.

To test the correctness of the implementation, a model with zero heat conduction ($\kappa_d = 0$) was compared with an analytic solution presented by Arter [14]. Dirichlet boundary conditions were applied that set $n(0) = n(1) = T(0) = T(1) = U(1) = 1$ and $u(0) = -1$. This test case, which is constructed with very smooth source terms, is presented in Figure 8.

A more realistic value for the conduction term $\kappa_0 = 200$ is described by Arter in [14]. To investigate the effect of this parameter on the steady state solution, κ_0 was varied between 0 and 2000 whilst using a momentum source term constructed from two sharp Gaussian distributions located near the two boundaries. The source terms for this investigation are presented in Figure 9a. As in the correctness test these momentum sources S^u oppose the flow of the plasma. In contrast to the

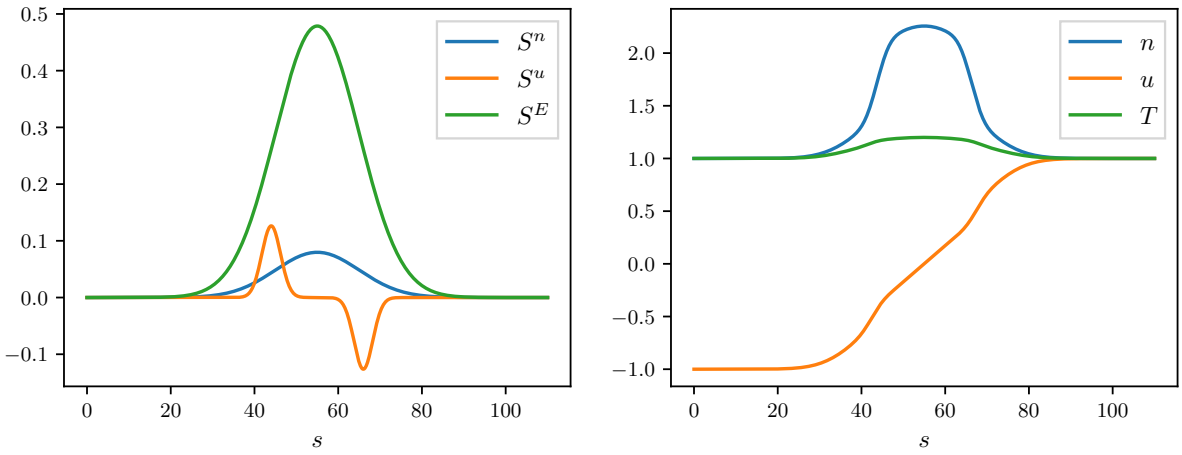


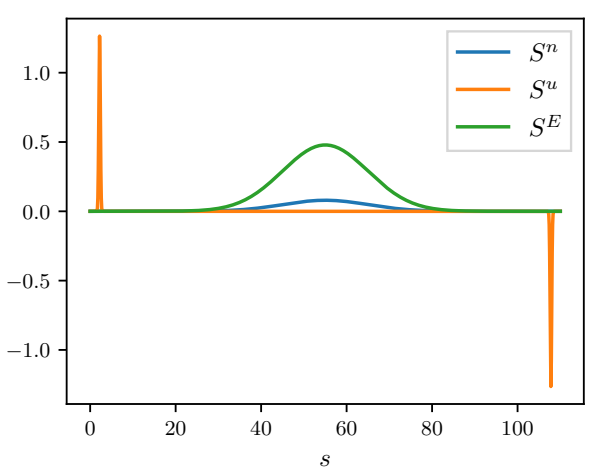
Figure 8: Source terms and computed steady state profiles for density, velocity and temperature for a zero heat conduction test case. Left: source terms for plasma density, plasma momentum and system energy. Right: steady state solution.

correctness test S^u is constructed with narrower Gaussian widths such that the resulting profiles feature higher gradients. The same Dirichlet boundary conditions used for the correctness were applied and the same spatial discretisation was used, i.e. mesh spacing $h = 0.1$ with first order Continuous Lagrange elements. The resulting steady state profiles from the sweep of κ_0 values is presented in Figure 9.

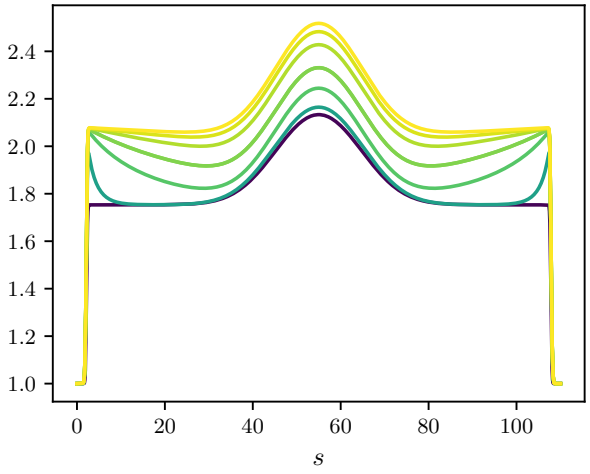
As expected the higher conduction coefficients, shown in lighter colours in Figure 9, result in smaller temperature variation over the domain and reduced temperature gradients near the boundary. The variation of κ_0 has minimal effect on the overall shape of the velocity profile but does slightly alter the shape at the inflection points near to the boundary. A similar effect is observed for the density profile where, at the inflection points, the intermediate values of κ_0 produce profiles which have a significant positive gradient before the turning point and then negative gradient when heading to the boundary. This is in contrast to the extreme values of κ_0 that produce density profiles with near-right-angled inflection points.

Our final section on the neutral-free model investigates the error of the computed steady state profiles as a function of mesh spacing h and polynomial order p. The density, momentum and energy sources and boundary conditions of our test case are identical to those applied in the sweep of κ_0 values and are presented in Figure 9a. The mesh refinement experiment applied first order Continuous Lagrange finite elements for all mesh resolutions and in the polynomial order refinement experiment the mesh spacing was fixed at $h = 0.1$. For this investigation κ_0 was given a fixed value of 200. The error in each of the test profiles was evaluated using a reference solution computed using a mesh spacing of $h = 2 \times 10^{-5}$ with Continuous Galerkin order-1 basis functions. This reference function space features approximately 5.5×10^6 DOFs for the simulation domain.

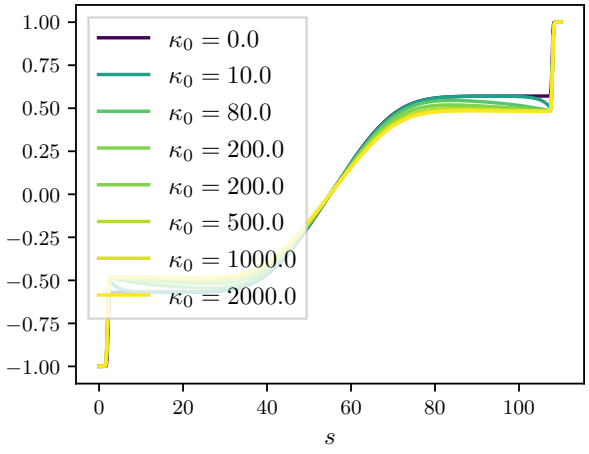
Each test profile was constructed by using the steady state solver to compute a new profile for the particular mesh spacing or polynomial order. In Figures 10 the L2 error in the computed solution is shown as a function of the mesh spacing h and polynomial order p respectively. The dashed black line is a reference that indicates an $\mathcal{O}(h^{p+1})$ rate of convergence.



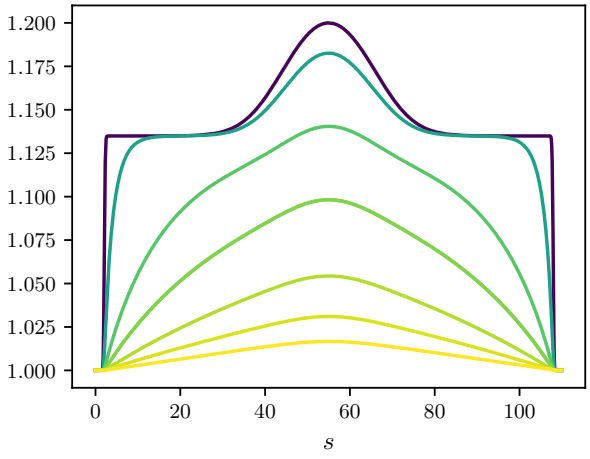
(a) Source terms for density, momentum and Energy equations.



(b) Density profiles.



(c) Velocity profiles.



(d) Temperature profiles.

Figure 9: Steady state density, velocity and temperature profiles for conduction terms κ_0 between 0 and 2000 for the presented source terms.

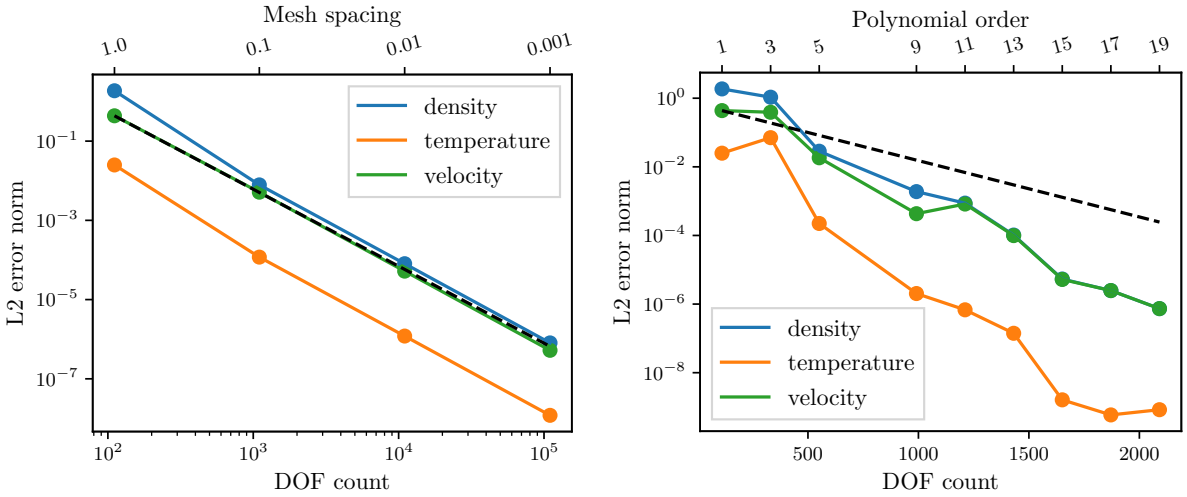


Figure 10: Left: L2 error as a function of mesh spacing h. Right: L2 error as a function of polynomial order p. Dashed black line indicates an error convergence behaviour of $\mathcal{O}(h^{p+1})$.

The error results demonstrate that, for this test case, the implementation attains a rate of converge that asymptotically behaves $\mathcal{O}(h^{p+1})$ both with mesh refinement and increasing polynomial order. Furthermore these convergence results were computed from profiles that exhibit significant gradients which are an expected feature in the SOL.

4.3 Conclusions and future work

It was demonstrated that finite element based implementations of 1D SOL models can be efficiently and correctly produced using the *Firedrake* framework. The implementation allows a user to easily modify the equations, discretization, and time stepping method, in contrast to a traditional monolithic code. These alterations can be performed without detailed knowledge of the inner workings of a finite element code. Furthermore the code generation approach of *Firedrake*, along with inbuilt use of libraries such as PETSc, mean that the produced implementation should be efficient.

With this implementation, the capability to produce density, temperature and velocity profiles that feature sharp spatial gradients similar in magnitude to those found for realistic source conditions was shown. This implementation provides a testbed to assess the impact of mesh refinement (h-adaptivity) and polynomial order refinement (p-adaptivity) on the error of steady state solutions. Future work should investigate inclusion of more realistic source terms and boundary conditions for the model and implementation. One avenue to produce realistic source terms is to study the addition of a neutral species to the model. This neutral species could interact with the existing plasma species via additional interaction terms that govern how charge and momentum are transferred between the two species. In the plasma physics community the model including a neutral species is considered a significantly harder problem that the neutral-free case.

5 Summary

This report has provided an overview and initial assessment of two of the NEPTUNE proxyapps, both of which utilize the spectral-hp method as implemented in the *Nektar++* framework. The author-supplied examples for each have been run successfully. In the case of *Nektar-Diffusion* the spectral convergence and the correctness of the anisotropic Laplacian solver have been demonstrated in simple cases.

An overview of 1D modelling using the flexible *Firedrake* framework was also provided, including plausible solutions to equations describing the physics of the scrape-off layer and a demonstration of spectral convergence. This study clears the path for the examination of more complicated models within the same framework.

Acknowledgement

The support of the UK Meteorological Office and Strategic Priorities Fund is acknowledged.

References

[1] D. Moxey et al. Nektar++ website. `https://www.nektar.info`, 2020. Accessed: June 2020.

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

[3] Nektar++ solver for Hasegawa-Wakatani equations. `https://github.com/ExCALIBUR-NEPTUNE/nektar-driftwave`, 2021. Accessed: September 2021.

[4] 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), December 2016.

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

[6] S. I. Braginskii. Transport processes in a plasma. *Reviews of plasma physics*, 1, 1965.

[7] D. Moxey et al. Nektar++: Enhancing the capability and application of high-fidelity spectral/hp element methods. *Computer Physics Communications*, 249, 2019.

[8] M. Wakatani and A. Hasegawa. A collisional drift wave description of plasma edge turbulence. *Physics of Fluids*, 27, 1986.

[9] Solving (modified) Hasegawa-Wakatani equations. `http://ammar-hakim.org/sj/je/je17/je17-hasegawa-wakatani.html`, 2021. Accessed: September 2021.

[10] B D Dudson, J Allen, T Body, B Chapman, C Lau, L Townley, D Moulton, J Harrison, and B Lipschultz. The role of particle, energy and momentum losses in 1d simulations of divertor detachment. *Plasma Physics and Controlled Fusion*, 61(6):065008, may 2019.

[11] E Havlíčková, W Fundamenski, F Subba, D Coster, M Wischmeier, and G Fishpool. Benchmarking of a 1d scrape-off layer code SOLF1d with SOLPS and its use in modelling long-legged divertors. *Plasma Physics and Controlled Fusion*, 55(6):065004, may 2013.

[12] E Havlíčková, W Fundamenski, V Naulin, A H Nielsen, R Zagorski, J Seidl, and J Horacek. Steady-state and time-dependent modelling of parallel transport in the scrape-off layer. *Plasma Physics and Controlled Fusion*, 2011.

[13] Satish Balay, William Gropp, Lois Curfman McInnes, and Barry F Smith. PETSc, the portable, extensible toolkit for scientific computation. *Argonne National Laboratory*, 2(17), 1998.

[14] Wayne Arter. Study of source terms in the SOLF1D edge code. Technical report, Eurofusion/CCFE, 2015. CCFE-DETACHMENT-RP2-Draft.