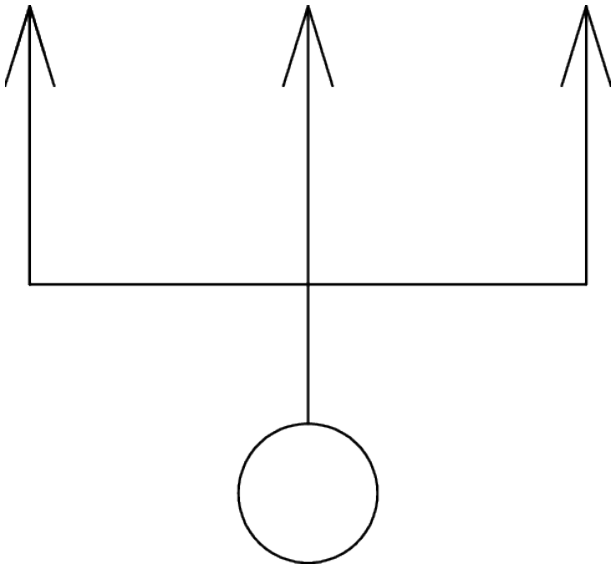
UK Atomic Energy Authority

ExCALIBUR Project NEPTUNE

Software Specification Website

Release 0.0.2

Abstract
Formal specification document to be used for a rational design of software. See the Development Plan **y2d34** , minutes of the workshop **y3re181**, and the development document **y3re314**.



Contents

Chapter 1

Program Name

1.1 Executive summary

The program addresses key plasma modelling issues for reactor design. This new software should become essential for designing optimal power-handling strategies at the tokamak first wall, both directly (via the simpler proxyapps) and indirectly by improving detailed physical understanding of the often turbulent, plasma-wall interaction.

This website is set out as described in **y2d34** to cover all aspects of the overall NEPTUNE package, as they emerge, beginning with the initial concept, advancing to detailed design of classes (objects) and interfaces, and ultimately producing documentation to ensure the software remains usable, maintainable and relevant for at least 30 years.

Further information

High-level project issues are covered by the project science plan **sciplan**; the NEPTUNE project is ExCALIBUR funded **exch+eswebsite**. Distribution and use of the software is covered by the extremely permissive MIT licence **MITlicense**, regarded as equivalent to the BSD3 licence. Collaboration is encouraged, and there are many additional benefits of community membership as set out below. Those interested in joining the community should email `neptune@ukaea.uk` to establish a dialogue.

Benefits of community membership

The principal benefit is access to what should ultimately become a powerful and comprehensive software package for modelling tokamak edge plasma using finite element and particle methods. In addition members will also gain

- Access to reports as the community produces them, in the access-controlled github site **xpndocswebsite**, subdirectories `reports` and `reports/ukaea_reports`. These directories already include educational material on

  - Finite elements
  - Surveys of current software
  - Surveys of current HPC machines and performance
  - Uncertainty Quantification
  - Aspects of software engineering, such as design patterns

- Rights to attend workshops and shape the NEPTUNE software, announced in the Slack channel.
- Right to attend project lectures on work performed by the community, and on relevant background material such as the spectral/hp element method, announced in the Slack channel.
- The `tex` subdirectory **xpntexwebsite** contains bibtex databases to aid report writing in subdirectory `bib` and graphics suitable for producing presentations in subdirectories `pics` and `png`.

The Slack channel is '#excalibur-neptune'. (The Slack communication software is downloadable from `https://slack.com/`).

(Note that access to most NEPTUNE reports is restricted to community members.)

Convention on use of IETF Keywords

The RFC2119 subset of the Internet Engineering Task Force (IETF) keywords **rfc2119** is used throughout the website, unless stated otherwise. Such usage implies specific meanings for "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" *when* the words are capitalised.

UKAEA REFERENCE AND APPROVAL SHEET

	Client Reference:	
	UKAEA Reference:	CD/EXCALIBUR-FMS/0054
	Issue:	1.00
	Date:	9th August 2022
Project Name: ExCALIBUR Fusion Modelling System		

	Name and Department	Signature	Date
Prepared By:	Wayne Arter Ed Threlfall Joseph Parker Will Saunders BD	N/A N/A N/A N/A	9th August 2022 9th August 2022 9th August 2022 9th August 2022
Reviewed By:	Rob Akers Advanced Computing Dept. Manager		9th August 2022
Approved By:	Rob Akers Advanced Computing Dept. Manager		9th August 2022

Chapter 2

Business Design

1. **CAPABILITIES**

   The NEPTUNE package will be capable of efficiently contributing 'actionable' results regarding the first wall to the reactor design process, specifically by speedily modelling the power deposited by the plasma

   - subject to uncertainty quantification (UQ)
   - using a modular suite of compatible components of minimal necessary complexity so as to ensure a workflow for rapid design, capable when needed of involving the latest high performance computers (HPC)

   In addition, the package will facilitate research into edge plasma physics by

   - its ease of use, providing a suitable DSL for 'high-level' usage in Python/Julia to enable intuitive additions to existing models or the incorporation of new models, whereby equations may be defined using LaTeX as explicit PDEs or as Lagrangians, and novel initial conditions imposed and new boundary conditions applied
   - its ease of modification and development, by providing a set of well-defined objects/classes for tokamak plasma physics
   - ease of incorporation of its component software into other physics packages
   - offering a careful automatic control of numerical error

2. **STRATEGIC FIT**

   National: The ExCALIBUR project is of national importance to UK government (represented by the BEIS dept.) to demonstrate how to produce software that can exploit all the latest, most powerful hardware for scientific computation. The Fusion Modelling System (FMS) is one of

ExCALIBUR's principal use cases, for which project NEPTUNE will explore efficient development of new software for the Exascale.

UKAEA: The software will facilitate world-leading R&D into fusion energy by UKAEA physicists and engineers, underpinned by external collaborators with a wide range of expertise. Its research contribution will be to improve detailed physical understanding of the often-turbulent plasma-wall interaction. Development of tokamak reactors will be facilitated if Exascale machines can be seamlessly integrated into the design workflow.

3. BUSINESS DRIVERS

Optimal power-handling at the wall will be critical for fusion reactors to be able to deliver sustainable fusion energy to the grid economically, if at all. Rapid exploration of parameter space is important to understand and optimise reactor designs, yet the existing software available to UK is dated so that it require years of experience to use well and is not suited to the latest HPC. Its replacement should remove a major handicap in the race to produce reactor designs.

4. ASSUMPTIONS

Funding of approx. £ 5 M over 5 years has been made available to UKAEA via the Strategic Priorities Fund under its ExCALIBUR programme to ready the UK for the Exascale era of computing. It is expected that further development of the NEPTUNE software will be funded at a similar if somewhat smaller level after 2025.

5. RISKS

Funding is via UK government and does not depend on the international situation.

6. IMPACTS

The new software should enable much faster iteration in respect of engineering design of the first wall of tokamak reactors. It should save much time and effort in the modelling of tokamak edge plasma physics. Generally, it should greatly reduce the training and computer time needed to obtain results compared to the existing software. There should also be benefits to UKAEA's wider relationships with the Eurofusion E-TASC programme and with ITER.

7. STAKEHOLDERS

The successful outcome of the project should be a plus for the UK, UKAEA and most employees. The only losers will be those UKAEA staff and contractors who have devoted often years of their working lives to the dated software that NEPTUNE is designed to replace, and will undergo a loss of status in consequence. However, their physics skills and understanding

will be valuable for guiding the NEPTUNE development to produce maximum benefits, hence they should soon recover position within UKAEA.

8. GOVERNANCE

The NEPTUNE project is overall governed by the UK governance known as PRINCE2 **prince2**, **prince2wiki**. which demands oversight by a local committee known as a project board. Finance is subject to the usual UKAEA procedures and controls. The NEPTUNE project is further subject to reporting to UK Met Office as part of wider ExCALIBUR activities, whence there is a second layer of PRINCE2 oversight.

The planned technical activities are outlined in the Science Plan **sciplan**, it and all major changes and refinements are subject to external refereeing, following the ExCALIBUR procedures drawn up by UK Met Office consistent with the demands of the UK Strategic Priorities Fund **SPF**. External procurements similarly follow the ExCALIBUR procedures drawn up by UK Met Office.

Chapter 3

Requirements Baseline

The Requirements Baseline (RB) expresses the user requirements for the software.

The largest input to the process, representing UKAEA Tokamak Science Department appears under Technical Specification Section **??**, so that it may be accompanied by a detailed response. Instead, there is presented an appreciation of the properties of the plasma edge in Section **??**.

The Departmental input, although it deals with other aspects of the specification, focusses more on the physical processes to be modelled and the approaches likely to be required. The following Section **??** records interactions with UKAEA Engineers, although a subsequent meeting clarified that the main demand for $\mathrm{NEPTUNE}$ at that time was that it be a modular, component-based design, equally suitable for integration into ANSYS OptisLang^{TM} as VECMAtk (with more detailed requirements likely to follow when the software became more developed). Thereafter the next Section **??** contains use cases, followed by Section **??** describing general requirements deducible from the use cases.

3.1 Physical properties of the edge plasma

The following scrape-off layer (SOL) parameters (including decay lengths) are for the L-mode scrape off layer in MAST **Mi13Expe**. For MAST, with the standard notation, $R = 1.6\,\mathrm{m}$, $B_T \approx 0.6\,\mathrm{T}$, $I_\phi \approx 400\,\mathrm{kA}$ and $a = 0.6\,\mathrm{m}$, values which imply that the poloidal field at the plasma edge $B_p \approx 0.1\,\mathrm{T}$. The main result of the paper **Mi13Expe** is that the decay length of the power deposition at midplane is $\lambda_q \approx 2\,\mathrm{cm}$ (range $1-3\,\mathrm{cm}$) and $P_{tot} \approx 350\,\mathrm{kW}$.

Unless stated otherwise, the derived length, timescales and speeds are derived from formulae and graphs in Wesson's book **wesson**. The derived quantities have been checked against SI formulae in ref **miyamoto**, and also compared with those listed in **Xu10Inte**. It is worth noting that although the latter table describes the SOL of JET (and in addition its separatrix and pedestal), JET values are typically within a factor of 2 of those for MAST.

Typical discharge

The edge values found experimentally are $T_e \approx 10\,\text{eV}$, $T_i \approx 20\,\text{eV}$, $n \approx 3 \times 10^{18}\,\text{m}^{-3}$. These imply that the Coulomb logarithm $\Lambda \approx 12.5$, and the flow speed $U_d \approx 10^5\,\text{ms}^{-1}$ may be estimated using $P_{tot} = 2\pi R \lambda_q n (T_e + T_i) U_d$. Sadly there appears to be no reliable determination of the neutral density n. (Note use of different font to distinguish neutral density from plasma density.)

Length scales

Debye length $\lambda_D \approx 10^{-5}\,\text{m}$.
Electron Larmor radius $\rho_{te} \approx 7 \times 10^{-5}\,\text{m}$.
$\rho_{ti} \approx 40\rho_{te} \approx 4\,\text{mm}$.
Mean free path for electrons $\lambda_{emfp} \approx 1\,\text{m}$ (parallel to field).

Time scales

Collision frequency (electrons with ions) $\nu_e \approx 3\,\text{MHz}$, $\tau_e \approx 3 \times 10^{-7}\,\text{s}$ **NRLpf07**.
Plasma frequency $f_{pe} \approx 15\,\text{GHz}$, $\tau_{pe} \approx 7 \times 10^{-11}\,\text{s}$.
$f_{pi} \approx 0.4\,\text{GHz}$, $\tau_{pi} \approx 3 \times 10^{-9}\,\text{s}$.
Cyclotron frequency based on $B_p = 0.1\,\text{T}$, $f_{ce} \approx 2.8\,\text{GHz}$, $\tau_{ce} \approx 4 \times 10^{-10}\,\text{s}$.
$f_{ci} \approx 1.4\,\text{MHz}$, $\tau_{ci} \approx 7 \times 10^{-7}\,\text{s}$.

Speeds

Electron thermal $c_{se} \approx 1.2 \times 10^6\,\text{m}s^{-1}$.
$c_{si} \approx 4 \times 10^4\,\text{m}s^{-1}$.
Alfven speed using B_T is $U_A \approx 10^7\,\text{m}s^{-1}$.

Collisionality parameter $\nu_c^* = \frac{q_e^4}{3m_p^2\epsilon_0^2} L_0 n_0 / C_0^4$

(note that $\frac{m_p^2\epsilon_0^2}{e^4} = \frac{1}{3}\, s^4 m^{-6}$.)
Taking $L_0 \approx 10\,\text{m}$, $n_0 = n$. Squared sound speed $C_0^2 = T_i(|e|/m_e)(m_e/m_i)$, $C_0 \approx 3 \times 10^4\,\text{m}s^{-1}$, implies
Collisionality parameter $\nu_c^* \approx 30$.
Peclet number $\approx 0.4\nu_c^* \approx 10$, but turbulent coefficients$\approx 1\,\text{m}^2\text{s}^{-1}$ will generally give a smaller value.

Resistive diffusion $\eta_d = 15\,m^2 s^{-1} \propto T_e^{-3/2}$.
(Note that there is a notational clash with η; fusion physics and astrophysics differ by a factor μ_0, so that $\eta_d = \eta(\text{fusion})/\mu_0$.)

Applicability of Fluid Models

A key requirement for fluid models is that collision times should be much less than the timescale of interest, which as the preceding subsections show is true, except in the case of τ_e, the electron-ion collision time, and for the electrons

more generally for dynamics along the field-lines. The ion gyroradius is also uncomfortably large compared to quantities of interest. Note that τ_e is the longest timescale in the classical picture of approach to a single fluid picture of plasma, other timescales, including the timescale for momentum to equilibrate, are shorter.

Single fluid MHD is widely used in astrophysics consistent with the eloquent advocacy by Priest and Forbes **priestforbes**. They point out that ideal MHD is consistent with the drift ordering, despite confusion caused by the easy possibility to misinterpret Hazeltine and Meiss **hazeltinemeiss** on the subject. (The point is that although MHD treats a faster timescale, it is valid on longer timescales, provided relevant smaller/slower terms are retained.) Moreover, SOL timescales involving filaments are fast, witnessed by the fact that the ion gyro-frequency is used as normalisation for electrostatic models in ref **Mi12Simu**, which from Section **??** is a not too dissimilar timescale $10^{-7}\,s$ to the Alfven timescale based on the poloidal field ($1\,\mathrm{cm}/10^6 \approx 10^{-8}\,\mathrm{s}$). Later, Freidberg **freidberg** showed that, at least in directions perpendicular to $\mathbf{B}$, the dynamical MHD equation applies to a more general 'guiding centre' plasma. The situation may be summarised by saying that complexity lies mostly in the transport (diffusive) terms as these attempt to account for low collisionality, finite Larmor radius (FLR) etc.

Perhaps fortunately, the terms predicted by kinetic theory will usually be small (except for the electrical conductivity) compared to the turbulent transport expected on the basis of both observation and theory of the SOL plasma. The simplest way to account for turbulence is to assume ad-hoc isotropic, uniform 'eddy' diffusivities in addition to the usual fluid advection terms. Lastly, in a simple extension of MHD, large τ_e is accounted for by allowing the electrons and ions to have different temperatures, consistent with observation. Effects due to the presence of a large neutral population in the SOL could well be significant, see next Section **??**. However neutrals are mainly expected to act as a sink of momentum and energy.

Effect of Neutrals

Formulae for a weakly ionised plasma are given in the Plasma Formulary**NRLpf07**. The collision cross-sections for electrons and ions respectively from ref **Ha91hydr** are $\sigma_s^{e|0} = 10^{-19}\,m^2$ and $\sigma_s^{i|0} = 4 \times 10^{-19}\,m^2$. Hence, the collision frequencies for electrons and ions respectively are

$$\nu_{e\mathsf{n}} = 1.2 \times 10^{-13}\mathsf{n}, \quad \nu_{i\mathsf{n}} = 2 \times 10^{-14}\mathsf{n} \tag{3.1}$$

where n is the neutral density. (Note use of different font to distinguish neutral density from plasma density.) If $\mathsf{n} = n$ is assumed, then the corresponding SOL collision times are

$$\tau_{e\mathsf{n}} = 3 \times 10^{-6}s, \quad \tau_{i\mathsf{n}} = 2 \times 10^{-5}s \tag{3.2}$$

so that the number of collisions experienced by a typical SOL ion before it hits a PFC is small. Nonetheless, since $1/m_e \gg 1/m_i$, $D_e \gg D_i$ and the diffusion coefficient for both electrons and ions is numerically large

$$D_A \approx (1 + \frac{T_e}{T_i})D_i \approx \frac{10^{23}}{\mathsf{n}} \tag{3.3}$$

The parallel electrical diffusivities are different, for electrons and ions respectively these are

$$\eta_{e\mathsf{n}\parallel} = 4\frac{\mathsf{n}}{n}, \quad \eta_{i\mathsf{n}\parallel} = 0.5\frac{\mathsf{n}}{n} \tag{3.4}$$

The implication from the formulae in ref **Le06emer** is that the value for $\eta_{e\parallel}$ combines additively with the usual Spitzer value in a more highly ionised plasma. Assuming $\mathsf{n} \approx n$, however the correction is seen to be an increase of 4 in $15\,m^2s^{-1}$, i.e. only about $25\,\%$.

Arber **Le06emer**, **Ar07Emer** further points out that according to the Formulary **NRLpf07**, in a weakly ionised plasma the conductivity is greatly reduced (and the magnetic diffusivity correspondingly enhanced) in directions normal to a strong magnetic field. Typically for Braginskii theory, the factor is x_e^2 for the perpendicular direction and x_e for the other direction, where

$$x_e = \frac{2\pi f_{ce}}{\nu_{e\mathsf{n}}} \approx \frac{8 \times 10^{23}}{\mathsf{n}} \tag{3.5}$$

For $\mathsf{n} = n = 3 \times 10^{18}$, these are huge increases. However it is worth noting that if the electromagnetic potential representation is invoked, so that

$$\mathbf{E} = -\nabla\Phi + \frac{\partial \mathbf{A}}{\partial t} \tag{3.6}$$

then, in the direction parallel to $\mathbf{B}$, neglecting the gradient of electric potential Φ

$$\frac{\partial A_\parallel}{\partial t} = \eta_{e\mathsf{n}\parallel} J_\parallel \tag{3.7}$$

Thus the enhanced diffusivities need not signify if this equation is used for magnetic field evolution, although applying a gauge condition on the potentials may become difficult in complicated 3-D topologies.

3.2 Engineering Requirements Baseline

Introduction

This set of requirements is based on four main sources, namely

- presentation by Chris Jones (CJ) at the NEPTUNE internal workshop on 16 December 2019

- points made in an email by Michael Kovari (MK) dated 19 December 2019
- interview with Zsolt Vizvary (ZV) on 20 December 2019
- use by author (WA) of SMARDDA-PFC software in tokamak reactor design 2016-2020

A significant part of CJ's talk concerned calculations of stress in 'contact' problems and in tokamak-relevant materials, particularly those to be used in and adjacent to the first wall. Mention was also made of nuclear heating and activation effects in the wall, and their effect on its strength. Since NEPTUNE is primarily intended as a plasma modelling tool, this material is neglected here except insofar as it has implications for NEPTUNE interfaces. Similarly ZV has as high priority, a capability to do magnetic equilibrium calculations so as to be able to calculate electromagnetic stresses in the wall. CJ emphasised that engineering design at UKAEA makes heavy use of the ANSYS^{TM} software for multiphysics calculations, and that since plasma effects including neutral beams and fast ions are not part of any standard finite element toolkit, particular consideration should be given to interfacing NEPTUNE to ANSYS macros and ACT (Python scripting for ANSYS). CJ stated that ANSYS had been implementing additional physics very slowly, on a timescale of decades, and there is anyway not a satisfactory licence model for using ANSYS on HPC, nor any to be expected soon.

Although the SMARDDA software takes as input general surface triangulations, which may therefore represent arbitrary surface topologies, SMARDDA-PFC calculates power deposition on the basis of a simple, empirical physical model of transport from the outer midplane. The errors in the model are frequently unquantifiable, given an absence of relevant experimental data, notably when 'small' limiters are proposed, where 'small' implies that many lines of magnetic field make several mid-plane passes before interacting with the limiter(s). The NEPTUNE software should provide a better physical model to enable (1) calculation of the midplane profile of power 'deposition' and associated parameters such as the e-folding length λ_q, when an exponential is fitted, and (2) an assessment of the accuracy of the whole SMARDDA surrogate model in a wide range of existing and novel configurations.

Overall Capabilities

Chris Jones has been led to dream of a
Whole-system full-physics digital twin available for real time in-silicon simulation and experimentation
but recognised that a more immediately realisable prospect was integration of plasma software into ANSYS Workbench^{TM}.

1. Simulations must be of known, improved accuracy, so that the performance of the built designs can be enhanced without sacrificing safety. Thus

the code and data structure should support mesh convergence studies, and be 'physics aware' as described in Section **??**.

2. The power load should be easily transferable to ANSYS for stress, heat transfer and other multi-physics analysis.

3. The software should be easy to couple to other physics packages produced within the fusion community such as RACLETTE and ERO for erosion calculations, LOCUST and SMARDDA-PFC for power deposition by particles, and neutronics software such as MCNP.

4. The software should be designed to support easy production of statistics from ensemble calculations, even when costs limit the size of the ensemble. The ensembles should be able to include different model selections as well as different parameter choices.

5. The software should be easy to use on HPC, eg. through cloud-based services, where it should be resilient to network bottlenecks. Its performance should scale well and not depend on the OS used.

6. The software should be able to exploit GPGPUs.

7. The software should use well-defined standard, open formats for both input and output of data.

8. The code should be easy to extend, without writing new code. Thus a user should be able to extend (at least virtually) a data structure to include a new parameter, and add a new physical effect.

9. The following aspects of the software should be open:

   (a) data structure requirements and documentation

   (b) code and documentation

   (c) test cases and their documentation

   (d) all results and their documentation

Physics Model

The software should be automatically 'physics aware', ie. it should

1. be able to test the physical assumptions and orderings used, perhaps by calculations at randomly placed points, and when this is impossible, to produce output to enable an independent person to check against a separate code.

2. be intelligent enough to switch to a simpler assumption when appropriate or when instructed to do so, for example by using classical transport coefficients when they apply.

3. be able to test the level of detail used. For example, the code (1) could carry out an ensemble calculation using a simplified model of radiative loss, and then check the results in a much shorter time against a fuller model using a sampling technique, (2) be able to identify automatically that a fully 3-D field calculation with say 12 or 18 discrete TF coils has produced a toroidally axisymmetric field in the vacuum vessel.

4. be able to simulate transient behaviour, being able to determine an initial approximate quasi-static solution, then depending on the length of the simulation relative to physical timescales of interest, perform either an implicit or an explicit calculation.

5. distinguish physical time from pseudo-time when relevant, eg. in an implicit calculation.

6. be able to handle incompatible timescales for bulk and local behaviour, eg. account for particle effects on overall flow.

Physics Capabilities

The software should be able to

1. compute the total power load on solid walls, due to plasma, fast ions and neutral beams
2. compute the production of impurity species by first wall melt and evaporation
3. compute heat conduction in first wall coatings in contact with the plasma
4. perform high frequency electrical and magnetic analysis, accounting for skin effects

Geometry

1. The software should be able to account for the effect of changes to geometry caused by radiation swelling, erosion and deposition due to plasma interaction, and corrosion.
2. The code should allow periodic toroidal boundary conditions.
3. The software should be able to handle 0-D, 1-D, 2-D and 3-D representations of the same plasma pulse, transferring between the different representations.
4. A related example concerns the recognition of field axisymmetry as in Section **??**.
5. Convexities in the surface, even sharp corners capable of causing singularities in the magnetic and stress fields, should be treatable by the software.

3.3 Use Cases

Use Cases: Tokamak edge physicist

They are early career, and to progress they need to build a professional reputation by publishing papers, supporting UKAEA's research programmes and supervising students. They are a competent developer and experienced HPC user, though they do not gain any credit directly from developing software.

In their research work, they study different models for the tokamak edge, and so require code flexibility and a user-friendly DSL to allow them to rapidly prototype different equation sets. This work would require quick iterations – perhaps 5 minute simulations performed on a desktop. They will also develop their own algorithms and add infrastructure to the code. While they will do this with an understanding of performance implications, they would expect to

perform these developments at a higher level that raw performance loops (but at a lower level than the physics model).

They would expect to contribute their changes back to a community repository, and also to benefit from changes that other code users have made. They would be involved in the community – perhaps raising issues, making and reviewing git pull requests, answering queries, and having input into future code releases – but would not be involved "project management" tasks, like maintaining the repository.

They will also value a user-friendly interface and active user community when it comes to working with their students. In this context it is valuable to have software that will run at a high level and produce sensible results without needing to specify the details of the implementation. This allows the student to learn about physical systems without simultaneously having to learn the details of numerical implementations. The active community allows their student to get support and ask (perhaps trivial) questions without being dependent on their supervisor.

Finally, in support of experiments, they will need to perform high-fidelity simulations of tokamaks. These will be highly computationally expensive, either because they are high-resolution simulations of specific shots, or because they are parameter scans or UQ campaigns. The simulations will be long-running, perhaps in the range of a week to a few months, on whichever HPC system that they have access to. The software must therefore be performance portable in order to facilitate high performance on a range of systems. The software also needs to be robust to numerical instabilities, hardware node failures, etc, as one may not have the resource allocation to repeat failed runs.

Use Cases: Engineers

As a thermomechanical engineer I:

- work with a large range of open source and proprietary codes which requires bindings to other tools, eg:
  - FMU
  - Python (Jupyter and regular)
  - OptiSLang
  - Twinbuilder
- work with CAD software to generate geometries which I want to propagate through my workflow.
- am interested in heat fluxes in all forms: from time and space averages to high resolution 3D time and data.
- need to be able iterate on designs quickly and in an automated way.

- am neither an HPC expert nor a plasma / tokamak physicist.

and I want to:

- know, given a sensible physics model provided by other experts, what the transient peak and average heat loads are on plasma facing components.
- not have to understand software dependencies and be able to install and run easily eg. "in the cloud".
- be able to configure the software to undertake parameter scans.
- have a handle on the sensitivity of the solution to the inputs and sources of error / uncertainty.
- be able to re-use the spatio-temporal heat fluxes as a model in more thermo-mechanical calculations. This means:
  - reading in the solution after the calculation.
  - using a fit to the data in the form of eg. a reduced order model of the heat fluxes, surrogates etc.
  - being able to export CAD geometries and import solutions back into engineering tools eg. ANSYS.
- have reproducible workflows to save and share with colleagues (eg. databases of inputs / outputs / config).
- export results flexibly to inter-operate with multiple surrogate frameworks.
- be in the loop with development process so it is possible to keep other workflows up to date.

This would mean I can:

- design components with colleagues within eg. STEP.
- make use of the NEPTUNE software in combination with proprietary tools that engineers know inside out.
- be insulated, to a sensible extent, from the complexities of the numerics, plasma physics and HPC.

Use Cases: Particle Specialists

As a particle specialist I:

- am very familiar with particle based methods.
- may not be familiar with FEM implementation details but have a working understanding of the approach.
- may not be familiar with low level languages.
- may not be familiar with HPC hardware and architectures.
- understand how to describe complex physical processes such as radiation, recombination, ionisation, charge exchange using both particle and FEM data.
- may not have applied UQ techniques before but may have an understanding of distributions/ensembles from statistical mechanics.

I want to:

- describe particle based operations both collectively and per particle, eg. :
  - creation and deletion of particles potentially from complex distributions.
  - computation with particle data - per particle and collectively.
  - identification of groups of particles.
  - representation of arbitrary per particle data.
- visualise particle and FEM data - snapshots and trajectories.
- create new finite element functions on appropriate function spaces.
- add source/sink terms to governing equations (solved with FEM).
- define particle source and sink regions using the simulation domain geometry.
- define regions of interest, eg. surfaces, as part of diagnostics.
- identify particles near surfaces/points/volumes of interest.
- create and use global data structures for computation, eg. diagnostics.
- represent particle data as FE functions:
  1. through pointwise projection.
  2. line integration over particle trajectory.

- use non-trivial functions in my loops, eg. erfc, gamma.
- define functions using expansion coefficients, eg. ionisation rate function approximated by an exponential expansion.
- evaluate these functions using both particle and FEM data.
- describe pairwise operations that implement physical processes.
- describe and sample from non-trivial statistical distributions.
- perform simulations in a reproducible manner.

So that I can:

- use particles as a kinetic description for plasma and neutrals.
- represent highly-collisional regimes by fluid approximations.
- describe plasma-neutral and plasma-plasma interactions.
- use abstractions/DSLs to write once, run anywhere as much as possible.
- experiment with models quickly and efficiently.
- perform ensemble computations and averages.
- perform UQ and verification.

Use Cases: Finite Element Background

I am a user with perhaps some grasp of plasma physics but with a more extensive knowledge of finite-element software (I might be an experienced user / developer of Nektar++). My background may be either physics or engineering; I may be a new recruit to the NEPTUNE team and needing to learn the code with a view to taking a future role as a NEPTUNE developer.

I need the interface / DSL to provide access to typical FEM parameters eg. choice of intra-element basis functions and their polynomial order, continuous / discontinuous Galerkin, choice of numerical flux, stabilization options; also whether diffusion and advection terms are explicit or implicit. In line with eg. Nektar++ I expect the choice of time-stepper to be largely "orthogonal" to most details mentioned above (the exception is explicit / implicit choice). I would like the option to specify the timestep in terms of the CFL number. In addition I require control over relevant meshing parameters eg. element spatial density and approximation order of any curvilinear elements. I would like the DSL to be able to generate a range of regular meshes internally (at least for trivial cases eg. boxes meshed with quads).

I should like some simple, physically-motivated canonical examples that might assist with learning plasma physics.

I expect the performance of the code to be at least commensurate with other FEM packages eg. Nektar++ and to remain so going forward (and obviously must be scalable to the latest hardware, which means foreseeably an efficient GPU implementation, supporting ideally NVidia, AMD, and Intel Xe / Ponte Vecchio).

I am unused to velocity-space effects. I would like the particles aspects of the code to be expressible, insofar as is possible, in FEM language: the conversion from discrete to continuum should ideally not be visible to me eg. converting particles to FEM forcing terms. Further to this, it would be good if a set of default particle parameters could be produced based on FEM parameters, as required (perhaps a reasonable value for the number of particles can be derived automatically based on FEM resolution).

If there is an issue from PIC compatibility (eg. constrained choice of basis functions), the DSL should make this clear in an explicit error message, plus hopefully advice how to remedy.

3.4 General Remarks

Particular, important general aspects of the use cases and Section **??** requirements may be as follows. Calculations may need be (re)started, perhaps from databases of calculations as envisaged by the IMAS development. Someone from the experimental side might want to specify input parameters by duplicating those of a particular say JET shot at a given time, using an database of experimental results. Physicists of either stamp (theoretical or experimental) will likely want compatibility with analysis software such as OMFIT **omfitwebsite** and tools to speed publication in the scientific literature of results obtained. An engineer may simply want to "change surface A to another design and repeat calculation".

Generally, minimising the number of new languages and systems people have to learn, especially in view of the need to attract people to the project and community, seems a good idea, so that a Domain Specific Language (DSL) should for example be based on one or more language(s) that are already well-known to many technical people, such as Julia, Python or LaTeX. Equally NEPTUNE software cannot be allowed to ossify, so the suggestion is everywhere to have a preferred option and an allowed option where this makes sense.

Chapter 4

Technical Specification

The Technical Specification (TS) contains the developer's response to the requirements baseline.

4.1 Response to Tokamak Science Division

Introduction

This section is Project NEPTUNE's response to Tokamak Science and MAST Upgrade Division's Requirements Specification document. This specification was received by Wayne Arter on 5/11/20 via Rob Akers and its authorship was confirmed to be Fulvio Militello and James Harrison at the NEPTUNE Project Board. It is intended that this document helps to frame expectations of the capabilities of NEPTUNE code, as well as highlighting challenges and areas of potential collaboration. Since the main focus herein is on aspects of the physical model, the next Section **??** goes into more detail regarding the software engineering needed to deliver the code successfully.

The physics requirements are summarised as:

> "The new UKAEA Exhaust code needs to be able to capture parallel and perpendicular transport of charged and neutral particles in 3D, full geometry and in a time dependent way. Turbulence should be self-consistently modelled, as well as energy transfer physics between charged particles, neutrals and photons (radiation). While perturbations need to be 3D, a minimal requirement for the code is that it can simulate realistic axisymmetric equilibrium configurations with complex topologies and wall designs. The aim of the code should be to:
>
> 1. Efficiently and reliably model exhaust in next generation experiments, like eg. MAST-U, JT60-SA, and especially ITER (the latter is a stringent requirement for the code).

2. Allow predictive exhaust capability for future reactor relevant machines like STEP or DEMO."

More detailed specifications are given in the following sections either as block quotations or as bold paragraph headers.

NEPTUNE Science Plan

The Science Plan for NEPTUNE **sciplan** is available online. The stated goal of the project is to:

> *"develop new algorithms, software and related e-Infrastructure that will result in the efficient use of current Petascale and future Exascale supercomputing hardware in order to*
>
> 1. *draw insights from ITER "Big Data"*
> 2. *to guide and optimise the design of the UK demonstration nuclear fusion power plant STEP and related fusion technology*
>
> *in the approach to the Exascale. The aims of the work are to deliver expertise in, and tools for, "in-silico" reactor interpretation and design."*

The Science Plan also describes the software development and theory development that is being and will be undertaken under NEPTUNE.

Software development. The aim of NEPTUNE software development is to provide a flexible framework for implementing different physical models in an Exascale-targeted manner. In particular, the project does *not* envisage a "NEPTUNE system of equations" so much as NEPTUNE providing the ability to solve a class of relevant equations, with models described relatively simply using a Domain-Specific Language (DSL). This flexibility enables the hierarchy of models of varying fidelity and computational costs. It also allows engagement from different classes of user with different levels of physics and software expertise.

Theory development. The above framework approach notwithstanding, NEPTUNE is also supporting theory development in two of its four work packages: FM-WP2 Plasma Multiphysics Model and FM-WP3 Neutral Gas and Impurity Model. These will develop two close coupled models, with FM-WP2 seeking to include kinetic effects in existing and new edge plasma models, and FM-WP3 developing particle-based models for describing the region outside and just inside the plasma (neutral atoms/molecules and partially ionised impurities).

Another work package, FM-WP1 Numerical Representation, is addressing related numerical issues, such as the accurate modelling of highly anisotropic

dynamics, the accurate representation of first wall geometry, and the numerical preservation of conservation laws from the underlying models.

As such the NEPTUNE plan is aligned with all the points in the summary requirements quoted above, though there are minor issues regarding the details as discussed in the following sections. It should be noted that the Science Plan outlines a five-year programme that explicitly requires user involvement for fuller development of surrogate models (such as turbulent friction) and to specify the detailed physics of ionisation and excitation reactions. It follows that development of many of the physics capabilities listed below will benefit greatly from a strong collaboration between Tokamak Science and Advanced Computing.

Overall Capabilities

Hierarchical approach with multiple models

"Hierarchical approach with multiple models, going from low fidelity (eg. laminar fluid and fluid-kinetic runs in 2D) to medium fidelity (eg. fluid runs with neutrals and turbulence multispecies, but with reduced number of species) and high fidelity (eg. full kinetic or hybrid kinetic/fluid runs with turbulence and multispecies approach)."

The plan is for NEPTUNE to provide a framework for solving relevant physics models, and a user-friendly Domain-Specific Language (DSL) for doing this. This allows UKAEA to develop its own physics models in a manner significantly independent of NEPTUNE development and code distribution, see Figure **??**. This is the same approach as currently used by UKAEA for BOUT++, where UKAEA has ownership of physics models in the STORM software, while other organisations have ownership of the BOUT++ framework.

Project NEPTUNE intends to provide a suite of examples of physics models as part of the code distribution. As addressed in §**??**, there is also development of a physics model under NEPTUNE that could be adaptive, displaying a range of fidelities within different regions of the domain during a single simulation.

Numerical efficiency

"Numerical efficiency obtained for example through scalability to large number of cores (given the expected computational resources, the code should be designed to take: $\sim$1 week for low fidelity parametric scans; $\lesssim$1 month for medium-high fidelity runs for advanced design; $\lesssim$3 months for high fidelity physics studies)."

Order of magnitude estimates for the cost of code execution (see Section **??**), indicate that these timings are possible but challenging.

The plan is to target Exascale simulations that can support high-fidelity physics simulations. Project NEPTUNE aims to do this in a performance

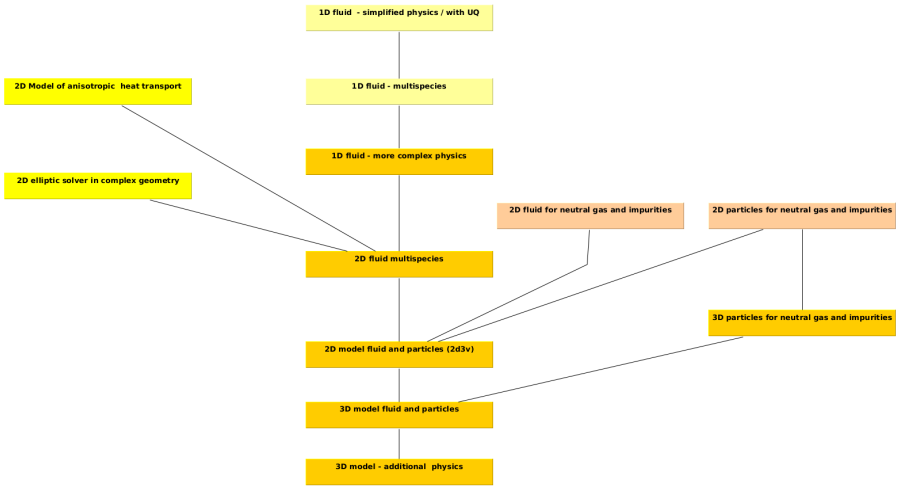


Figure 4.1: Development of NEPTUNE software via a sequence of proxyapps.

portable fashion using abstraction layers to separate the physics from the computation details. However, the challenges of Exascale mean that a degree of specialisation of the code towards available Exascale machines is expected. At some point this will probably entail trade-offs that are detrimental to performance on smaller machines. Despite this, we still anticipate that the software should be reasonably performant for small-scale runs.

Stability of the code

"Focus on stability of the code, using (preferably) unconditionally stable numerical schemes and capability to diagnose and re-start failed runs. Ensure a small failed simulation rate for common configurations."

The focus will be on accuracy rather than stability. An accurate solution will be a stable one, whereas the converse is not true. The use of bifurcation tracking techniques is expected to help with numerical stability. Moreover, the use of ensemble techniques (as part of uncertainty quantification) should provide information on stability as a function of parameters, as well as making it likely that at least a subset of calculations produces usable results.

In addition, there are some planned technical solutions for diagnosing problems with runs. The simplest functionality is to allow users to change parameters when restarting a job using checkpoint files. The code can also be made to automatically check input files (to catch user misspellings), and to output a file containing the parameters that were actually used in a simulation (in case some were accidentally overwritten). Simulations may also be given a Universally Unique Identifier (UUID) to allow provenance tracking of simulations.

Exhaust physics modelling capability

"Provide capability to model exhaust physics all the way from sheath limited to strongly detached regimes."

This capability follows from (1) the implemented physics models/boundary conditions, and (2) the lengthscales the code can resolve in simulations of feasible run times. Project NEPTUNE anticipates that this will be feasible. See §**??** for further details for the physical models.

Modern software design

"Modern software design with modular approach (independent and efficient libraries)."

Project NEPTUNE is certainly adopting a modular approach to software design. This is crucial for enabling many of the desired features, as noted below. The use of reliable third-party libraries is also vital for enabling a small team of developers to leverage the work of others, and to allow code flexibility and ease of prototyping.

Ability to integrate with other codes (eg. with IMAS).

This feature is anticipated, and will be facilitated by writing the code in object-oriented C++.

Integration with IMAS (and other data formats/standards) can be achieved by writing a module to translate between NEPTUNE's internal data structures and IMAS format. NEPTUNE developers are involved in TSVV software which will also integrate with IMAS, so will have experience in this area.

Modern visualisation tools

Integration with modern visualisation tools is not specifically addressed in the science plan. However, this is enabled by using standard data formats such as netCDF/HDF5. Auxiliary tools (similar to BOUT++'s xBOUT library) could also be developed.

The issue of *in situ* visualisation will also be important at the Exascale, with the need to interrogate large quantities of data without moving it, perhaps during a simulation. Project NEPTUNE does not have specific plans for enabling this. However, the need for this will be widespread, and we expect third-party tools/libraries to become available to support this, particularly in C++.

Accessibility (output easily catalogued and interrogated big data)

Given the constraint of producing Exascale volumes of data, it is expected that NEPTUNE will default to using the Met Office practice of only saving the files

necessary for repeating a simulation, rather than full outputs. It is intended that key aspects will be captured by surrogates, descriptions of which will be saved and indexed. Users however will be able to choose to output more data and to define custom diagnostics. A modular approach to software design and the use of standard libraries will ease the implementation of these.

In time, there will likely be a move towards creating simulation databases in preference to repeating expensive simulations. Such projects are nascent, but we have collaborators who are working in this area. Again, the modular approach and use of standard data structures will enable interfacing with such databases as the technology matures.

Version control and user support.

Version control is a fundamental part of modern software development, and will certainly be used in NEPTUNE. More generally, the project will use best software practices, including protected branches, peer reviewed pull requests, issue trackers and automated testing. The main code contributors are familiar with such practices; we have heavy involvement from the projects BOUT++ and Nektar++ which have very high quality software practices. Project NEPTUNE also have involvement from UKAEA RSEs.

User support will be provided through issue trackers, a project Slack channel, and where relevant, through training sessions. Project NEPTUNE anticipates supporting several classes of user, from those with limited software knowledge, to users with appreciation of different numerical/algorithm choices, through to numerical/software specialists who might edit the code on a granular level. Support for such a wide range of uses is enabled by a highly modular separation-of-concerns approach.

Physics model

General requirements

> "Equations need to be applicable to arbitrary aspect ratio devices."

The equations employed will not generically make the assumption of small inverse aspect ratio.

> The collision operators between plasma components, plasma and impurities and plasma and neutrals have to be sufficiently accurate to properly describe the radiation generated and the energy transfer between species.

These operators may be included in the model being derived in FM-WP2, but at this stage simpler model operators are being used.

> "Photon opacity effects need to be included in the model."

The treatment of effects due to radiation are discussed in Annex A of ref **pappeqs2** ("the equations document"). Initial implementation will account only for usage (1) *the calculation of source/loss terms*, with usage (2) *the production of synthetic spectra*, depending very much on input from experimentalists, as anticipated in the science plan **sciplan**.

> "The equations need to be capable of dealing with multiple species in non-trace amounts (at least D, T, He and seeding impurities)."

The models being derived support distribution functions for multiple species.

> "The equations should not rely on the Boussinesq approximation."

The model derived in FM-WP2 does not use the Boussinesq approximation. Technically, this means NEPTUNE will need to be able to invert three-dimensional elliptic operators with non-constant coefficients. This capability will be available.

> "The code should evolve both the electron and ion energy (temperature)."

This is the case with the FM-WP2 model.

> "It would be acceptable to have only axisymmetric equilibria, potentially corrected through small perturbations."

This is answered by the discussion in Section **??**.

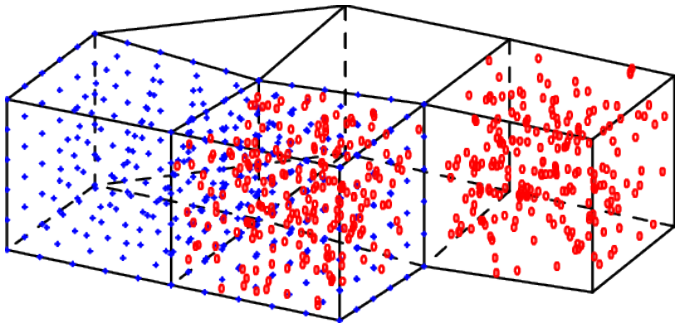


Figure 4.2: NEPTUNE software will use finite elements and particles to represent different atomic species as needed.

High fidelity simulations

> "In high fidelity simulations, the kinetic/fluid transition for both plasma and neutrals has to be properly captured, potentially using a multi-region approach exploiting different models in different parts of the machine could be considered (for both neutrals and plasma); Boundary conditions need to consider improved sheath physics (collisional & shallow angles); Ideally, the model should be able to treat de-magnetized ions in the divertor region; Electromagnetic effects should be included, but a perturbation approach would be acceptable."

The Science Plan states that *"FM-WP2 and FM-WP3 will concentrate upon development of the two close coupled models of the* NEPTUNE *programme, specifically FM-WP2 around the inclusion of kinetic effects into existing and new edge plasma models, and FM-WP3 of particle based models for describing the region outside and just inside the plasma (neutral atoms/molecules and partially ionized impurities)."* See Figure **??**.

In FM-WP2, a novel "moment-based" approach is being developed (for both plasma species and neutrals). This approach is a variant on gyrokinetics (though at present, the derivation exists only for drift kinetics). Instead of the stationary background Maxwellian of the usual gyrokinetics, this approach uses a dynamically-varying background Maxwellian, where the particle density, bulk velocity and temperatures are allowed to vary with time. The result is a hybrid approach, where the software evolves both a fluid system and a modified kinetic system. One of the benefits of this approach is that it gives a natural means to capture the fluid/kinetic transition, by switching between the fluid+kinetic and fluid-only descriptions. Moreover this approach allows the software to detect which of the fluid and kinetic descriptions is valid in any region based on the properties of the modified distribution function, and therefore to automatically evolve the relevant model.

The favourable properties of this approach are valuable, but as with any novel approach there is an inherent risk of the scheme not working. Thus as described in ref **pappeqs2**, in the event that this approach is not feasible, the kinetic calculations will use a Particle-in-Cell (PIC) approach. While full orbit PIC is potentially extremely expensive relative to gyrokinetics, it has been very well-studied both in terms of theory and numerical implementation. Moreover the issue of how to treat the transition from fluid to a particles in a coupled model is well-understood, at least in classical fluid dynamics.

Relevant sheath physics boundary conditions are being derived as part of this work, including shallow angles and collisions. Ions which exit the domain re-enter as neutrals with the Knudsen cosine distribution.

While the current derivation is electrostatic, there is nothing which precludes the derivation of an electromagnetic model.

Neutrals and impurities

Different levels of multispecies capabilities should be considered. The framework nature of the code means this will be the case.

In medium to high fidelity models, neutrals and heavy species should be treated fully kinetically (the former have potentially low collisionality and they are not bound by the magnetic field, the latter have a large Larmor radius). Development of a neutral gas and impurity model is ongoing under FM-WP3. Page 7 of the Science Plan states: *"Kinetic levels of complexity are . . . necessary . . . for modelling the burning plasma regime, due to the inherent uncertainty in the fluid codes. The plasma in a fusion reactor may well behave significantly differently to plasma in existing devices because it will in general contain two main ionic species (Deuterium and Tritium), neutral fuel particles and ionised Helium ash (or alpha particles), as well as impurity ions originating from the wall."* That is, kinetic models will be derived and implemented, though it is hoped that this will facilitate the development of fluid models.

In high fidelity models, dynamical neutral evolution on the turbulence time scale would be preferable, if possible. High-fidelity simulations will use a kinetic model for neutrals where appropriate.

Ability to model pumps (either directly or via pumping surfaces) should be included. Pellets: the code should enable simulation of, or coupling to models that can model, pellet fuelling. Models for these aspects are not included in the Science Plan. As we will use an unstructured mesh, it will be possible to describe arbitrary pumping surfaces.

The role of sources and sinks is recognised to be crucial in NEPTUNE physics so that modelling of pellets as a source will be possible. The tool

CWIPI (Coupling With Interpolation Parallel Interface) may be used for coupling to more complicated pellet models.

Physics capabilities

"The code needs to be able to give reliable predictions of:

- Divertor loads:
  - Reliable calculation of upstream particle and heat flux profiles: proper drift physics and upstream turbulence;
  - Divertor turbulence: turbulence spreading in the divertor region, effect of magnetic shear next to the X-point(s) to understand upstream/downstream connection;
  - Multifluid capacity to model radiation and detachment physics;
  - Reliable calculation of the electric field (collisionless physics near the separatrix, proper reflection of neutrals at the target);
  - Able to capture in/out and top/down asymmetries (geometry, drifts, radiation and detachment)
- Wall loads:
  - Should be able to predict filamentary transport and role of hot ion confinement (wall erosion)
  - Should calculate radiation and neutral loads (may require dedicated modules, same for divertor loads);
- Impurity transport, pumping and fueling:
  - Ability to track low (He Be), medium (C, N, Ne) and high Z impurities (W, Ar, Xe) with multiple charge states;
  - Reliable kinetic modelling of neutrals to assess fueling and pumping capacity;
  - Ability to handle non-trace species (D, T, He, seeded species), which requires new closures for the equations (if fluid, beyond Braginskii $\Longrightarrow$ Zhdanov or better);
  - Accurate model of friction forces (turbulence + neoclassical on open field lines);
  - In high fidelity models, the code should be able to simulate localized gas puffs, via injection of neutral particles.
  - Dust the code should enable coupling to codes to simulate dust generation, transport and ablation, via exchange of fluxes and sources."

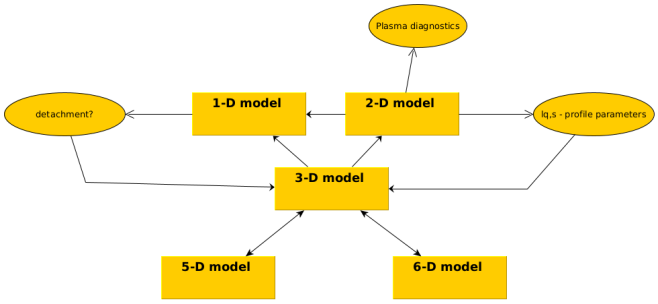


Figure 4.3: Project software will integrate a range of models of different complexity, here indicated by their dimensionality.

Physics models. Many points in this section relate to the physics model. As we have alluded to above, the actual physics model used is the responsibility of the user; so, for example, it will necessary for users to derive an accurate model for turbulent friction. Moreover, as NEPTUNE will initially be an edge code, only electromagnetic radiation from the edge plasma will be calculated, and an additional model will be required for radiation for the core plasma. Nonetheless, we plan to facilitate models which have the listed properties. In particular, our meshing approach will allow accurate representation of the walls and divertor. Thus powerful and flexible source/sink models may be used, for both volumes and surfaces.

NEPTUNE code will also be capable of representing scales on which filamentary transport has been calculated to occur. The code will only produce fluxes however, and it will be up to the user to calculate wall erosion.

Validation, Verification and Uncertainty Quantification (VVUQ). Naturally, when the solution of a problem is unknown, it is difficult to assess whether the provided answer is reliable as requested in the first line of the current section. Therefore we are integrating NEPTUNE with a suite of tools for VVUQ to enable validation of code outputs against experimental results, and provide a error bars for code outputs due to the uncertainty in the code inputs. As allowed by error estimates, models of different complexity will be used to predict key properties of the SOL, see Figure **??**.

Geometry

"Ability to handle complex topologies and novel geometries:

- Capacity to treat different topologies with multiple X-points (>2), possibly in a dynamic way, or at least implemented in a way that does not preclude time-varying equilibria being modelled in future;
- Ability to handle singularities in the grid (null points) or to develop accurate scheme(s) that do not have singularities;
- Ability to interface with 3D CAD designs of the machine;
- Capability to handle conformal grids that go all the way to the wall for both neutrals and plasma;
- Capability to treat subdivertor structures (only neutrals)."

NEPTUNE will use a spectral/hp finite element approach. This allows considerable flexibility in meshing, with elemental tetrahedra, hexahedra or prisms being used to describe complex geometries. In addition to refinement of the elemental grid sizes (h-adaptivity), the order of the piecewise polynomial basis functions used on each element may also be changed (p-adaptivity). The production of finite element griddings conforming to CAD descriptions is standard for virtually all meshing packages. Project NEPTUNE will specially aim to use the small number of packages that may produce finite elements with curved edges and faces to exploit fully the higher order basis. This gives a framework which can achieve spectral convergence on complicated, conforming grids, while still being able to handle discontinuities in the solution.

Should it prove necessary to produce a meshing conforming to surfaces of an equilibrium magnetic field, then because the finite element approach does not require a structured grid, it should be able to treat arbitrary geometries and topologies, in particular including multiple X-points. The approach could be extended by further use of adaptive meshing to allow for time-varying equilibria.

4.A Order-of-magnitude estimates for tokamak edge modelling

Suppose plasma number density $n \approx 10^{18}$ m^{-3}. Order of magnitude dimensions are $L \approx 0.1$ m for SOL thickness, reactor minor and major radii say $a = 3$ m and $R_0 = 10$ m, so the volume of SOL $\approx 4\pi^2 a R_0 L \approx 100$ m^3. It follows that the total number of electrons $\approx 10^{20}$.

The shortest timescale is inverse eB/m_e, the electron cyclotron frequency, where the ratio of charge to mass for the electron is $|e|/m_e = 1.76 \times 10^{11}$ C kg^{-1}, and $B \approx 10$ T, so $\tau_{Be} \approx 10^{-12}$ s. Hence the number of particle-steps to evolve 1 s of physical time is $10^{20+12+1}$ (assuming of order 10π timesteps per

orbit), which assuming 1000 flop per update, needs a total of 10^{36} flop. Thus to complete a computation in 1 s on Exascale machine, only one 1 particle-step in 10^{18} is allowed. This implies for example only 100 superparticles in the volume can be used supposing each has a weight 10^{18}, which is unlikely to be adequate because electrostatic and other effects will produce a noise level that swamps any physical effects. However if 3 or 4 months (approx. 10^7 s) are allowed, then a calculation with 10^9 particles may be performed if memory-access bandwidth constraints can be satisfied, when the noise levels may be manageable.

The situation may be shown to be a million times easier for neutrals considered separately, assuming neutral density $n_0 = n$, particle mass m_p and temperature $T_0 = 10\,\text{eV}$, for a timescale of $\tau_0 = L_{mfp}/v_0$ where the mean free path $L_{mfp} \approx L$ and the neutral speed typically is v_0. For then $v_0 = \sqrt{|e|T_0/m_p} = \sqrt{|e|/m_e} \cdot \sqrt{m_e/m_p}\sqrt{T_0}$, ie. $v_0 \approx 4 \times 10^5 \times (1/40) \times \sqrt{10} \approx 3 \times 10^4$. Thus $\tau_0 \approx 10^{-6}$ s, ie. a million times longer than the electron cyclotron timescale.

Suppose a fluid model is employed instead, ie. the electron distribution is represented by its first 3 moments. If the electron temperature $T_e \approx 10\,\text{eV}$, then the thermal speed $V_{Te} \approx 40 v_0 \approx 10^6$ m s^{-1}. Supposing the SOL to be sampled at a uniform 1 mm interval, then the number of sample-points $\approx 10^{11}$, and the timestep for an explicit scheme $\approx (10^{-3}/10^6) \approx 10^{-9}$ s, so the number of sample-point updates is 10^{11+9}. Assuming 1000 flop each update, this means one second of physical time needs 10^{23} flop or 10^5 s≈ 1 d of Exascale machine time, if memory-access bandwidth constraints can be satisfied. If an implicit scheme is used to simulate plasma ions as a fluid on a drift type timescale$\approx L/V_{Ti}$, then possibly the timestep $\tau_i \approx \tau_0 \approx 10^{-6}$ s, ie. a thousand times larger, and calculations lasting only a few minutes might suffice.

Another way of looking at this is to suppose that numerical problem is D-dimensional, 1000 flop are needed for each sample update and N_D samples per spatial dimension and N_D^2 time samples are allowed. Then to update such a model in 1 s requires $N_D^{D+2} \approx 10^{15}$. Thus if $D = 3$, $N_3 \approx 1\,000$, and $N_5 \approx 100$. It seems that accuracy controlled, unstructured, implicit fluid models should be possible, although for the more complex models 1000 flop per update may be an underestimate by orders of magnitude.

4.2 Software Engineering Response

The above scientific response Section **??** indicates that the NEPTUNE suite will have the capability to describe the tokamak edge in a comprehensive set of levels of detail using a large range of possibly heterogeneous computer hardware, will be straightforwardly modifiable to include additional physical effects, and will aim to include under all circumstances, the level of error in computed results. Evidently, eveloping the suite represents a major challenge to current software engineering practices thanks to its scientific difficulty as indicated in Section **??**,

and perhaps less obviously, its scientific *complexity* due to the need to treat large numbers of atomic and molecular species, descending to the level of isotopes with a range of charge states and electronic excitations.

The last makes NEPTUNE a significantly harder development than the BOUT++ code discussed in Section **??**. The difficulty motivated studies of software engineering practices outside as well as in the scientific context. The studies of scientific work are summarised in reports concerning frameworks, scientific UQ etc. Ideas from these studies are combined with non-technical works such as Hewitt **hewitt** and Sommerville **sommerville** and summarised in the report **y2d34**.

The outcome is the present report and website, which augment the procedures of Dudson and BOUT++ collaborators **y3re314** to govern the development of NEPTUNE. Management aspects of the development are treated in Section **??** and operational aspects in Section **??**. The remainder of this section focusses on the generic details of a software implementation, the so-called non-functional aspects, with BOUT++ instructions augmented to allow for the greater complexity of NEPTUNE, following the D3.3 reports **y2d33**, here with Section **??** adding a way to handle a multitude of variable names to accommodate the much increased number of physical variables.

Following D3.3 **y2d33**, Section **??** discusses high-level constraints on the structure of software. where the concept of division into packages and modules (which may represent libraries) is promoted. Section **??** explores the implications of these constraints for NEPTUNE. At the opposite extreme to Section **??** and Section **??**, Section **??** describes the desirable contents for a single module, and Section **??** discusses the question of what best to output when developing software for the Exascale.

General Considerations

It is important that individual units of code are manageable and the overall structure is comprehensible, so that developers and users can navigate the codebase and determine where new work is to be located. This simple consideration implies that NEPTUNE software should be divided into units that it will be convenient to refer to as modules, ie. sections of code containing everything necessary to perform one (and only one) aspect of the desired functionality.

The suggested content of a module describing a single class in the UML sense (see Section **??**, Table **??**) implies a minimum of 13 methods described by separate subroutines/functions, with examples extending up to 50, although the latter is there regarded as an excessively large number of methods. Many small software libraries also fall into this size range. Supposing that each subroutine is of a length to fit within one computer window, ie. up to about 60 lines, the desirable maximum length of a well-designed module file is $30 \times 60 = 1\,800$ lines which is a manageable size of file given editing software that remembers on restart, the last line accessed (so that it possible to return immediately to a

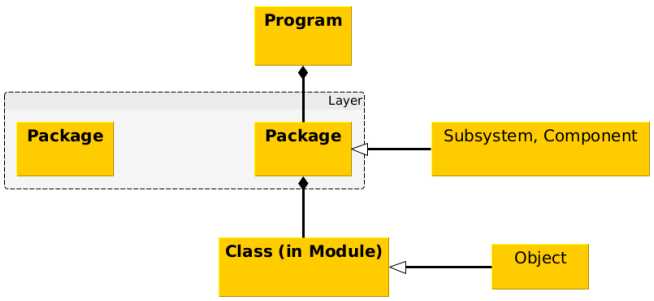


Figure 4.4: Sketch of relationship between units of large code.

particular subroutine). The magnitude of the D3.3 exercise follows from the fact that comparable software packages probably of somewhat lesser complexity than NEPTUNE written in high level languages such as C++ range in size up to $1\,000\,000$ lines. Clearly 400 separate modules is too unwieldy, and there is a need to organise further into packages which might contain in turn 10-50 modules. As a way of providing further indication to developers, it is helpful to talk of packages as being arranged into layers, as discussed in ref **y2re333**, see Figure **??**. Then, as prefigured in ref **y2re333**, it should be possible to encapsulate the necessary complexity in one, albeit large diagram.

Considerations for Scientific Software

Structural Considerations

In both refs **y1re331**, **y2re332**, a figure from Dubey et al **Du16Idea** is reproduced that illustrates how scientific software may be developed by beginning with an “infrastructure" capability into which initially exploratory scientific software is integrated as its worth is established. Unfortunately for NEPTUNE, it is not clear initially what aspects of the infrastructure will be durable and stable, although once the software is more mature, Dubey et al's methodology appears attractive.

The literature referenced in ref **y2re333** indicates that scientific software should be produced by aggregation, but is less helpful as to what is to be aggregated, ie. the modular decomposition as to what should constitute a single module etc.

Since the NEPTUNE development will proceed as a series of proxyapps, there is a chance to refine and redefine an initial modular decomposition with each successive proxyapp, recording the results as a UML structure diagram. When generating the corresponding sequence diagram (ie. procedural description), the proxyapp-based units should be preserved to facilitate the use of the simpler ones as surrogates for the later more complex proxyapps.

To help understand the aggregation of the software, it should be layered in

the obvious manner, with the higher layers corresponding to greater numbers of aggregated objects. It is also expected to be useful to classify the modules. The modules should be arranged into a relatively small number of packages according to, eg. whether they treat particle generation, matrix coefficient calculation, the main matrix solution, or visualisation.

Interactions between Modules

Concerning logical interconnections between modules, the use of a directed, acyclic graph (DAG) structure might be thought mandatory, particularly to process the input data in order to specify coherently the construction of the elements of the solution matrix. However, as prefigured in ref **y2re333** for the gyrokinetics code GS2, the tightly coupled nature of the central edge physics problem means that input is more about gathering *all* the data, for only at that point can fields be computed and only after that may matrix coefficients be computed.

(TODO) "Information hiding principle" eg. input data module "hides" information about how this is done from the calculation module: it just passes along parameters in an agreed format. Can extend this idea on to every level - eg. Parnas advocates hiding the hardware from the code - cf. performance portability. Such modularity enables work to be efficiently assigned to a programmer or group.

Design of a Module

Notation

UML nomenclature is preferred herein, for which Table **??** provides a limited translation into C++ and Object Fortran.

Table 4.1: UML interpreted as Object Fortran and C++

UML	Fortran 2003	C++
Class	Derived type	Class
Part	-	Component
Attribute	Component	Data member
Method	Type-bound procedure	Virtual Member function
Feature	Component and Type-bound procedure	Data and Virtual Member function
Structured class	Extended type	Subclass
Specialisation	Class	Dynamic Polymorphism
Generalisation	Generic interface	Function overloading

Further insight into UML terminology may be gained from the description of the patterns in refs **y2re332**, **y2re333**.

Module design

The focus herein is the structure within a module. Specifically, the module describes a single class or object (strictly speaking in UML terms, objects are instances of a class), which is fundamental in that it is defined without use of aggregation. Tthe software - consistent with Clerman and Spector **clermanspector** - that is promoted by Arter et al **fprog** for an object-oriented language, recognises two sizes of fundamental class and it is easier to start by considering the smaller, denoted smallobj_m.

Module smallobj_m does not have a separate attributes file, but will typically still use or access three or more yet more fundamental classes, namely

- log_m for logging errors or warnings in code execution, and checkpointing values of selected variables
- const_numphys_h for numerical values of important mathematical and physical constants relevant to plasma physics
- const_kind_m to specify precision of representation of real and integer values, together with formatting to be used for their output (in fact contains no executable code)
- date_time_m to return the date and time in either a verbose or minimal format
- misc_m to form miscellaneous operations found to be common to many modules

In outline, the methods or operations associated with `smallobj` allow data to be read from a named file, used in the construction of an object, and output to disc file. The file is given a numeric identifier `ninso` (file unit) when it is opened. Data used to construct the object forms a single `namelist` block, viz. a list of arbitrary code variables that may (or not) be assigned values in the input file using a attribute-value notation. Namelist variables should have long names that promote a good user interface, and be given default values in case they are not input. The style encourages checking that inputs have acceptable values, for example lie in expected ranges, but there is no equivalent of eg. the Cerberus Python data validation package **cerberuswebsite**, users must explicitly code checks and calls to log_m if the values are questionable or erroneous. (It is hoped that this can be automated based on a LaTeX table describing input symbols. After successful checks, the values in the namelist are copied into the data type sonumerics_t, whence it is supposed that a single subroutine then instantiates the smallobj_t object. Another routine performs output of this object, either to a directly specified file unit, or to the standard output by default, in the simple text format of a variable name followed by its value on the following line. As might be expected, there are also routines

to 'delete' the object, ie. to deallocate any constituent arrays, and to close the input file. The precise list of member functions (or methods in the UML nomenclature) is

- initfile, open input file
- readcon, read data from input file
- generic, generic subroutine to instantiate object
- write, write out object to standard output, or to file opened by another object
- delete, delete object
- close, close input file

Module bigobj_m has a separate file for its attributes (*aka* namespace), which it will normally still use or access like the yet more fundamental objects listed for smallobj_m. However the data types defined in bigobj_h are of the same kind as those in smallobj_m, viz. bonumerics_t to hold input data which is used to define bigobj_t by calling the `solve` subroutine (rather than the subroutine `generic` in the case of smallobj_m). Apart from this last exception, the methods in bigobj_m are a superset of those in smallobj_m. The additional methods recognise that instantiation may involve more than one routine and in particular may involve use of a specially defined function `fn`, demonstrating the STRATEGY Pattern or possibly TEMPLATE Pattern. This function may be determined by a formula identified in the input, giving the option for a developer or determined user of adding their own code to define a new function. The range of allowable outputs is much extended. Thus there is a separate routine provided for output to the log file using what will probably be a lengthy list of calls to log_value_m. More likely to be useful is a routine to open an .out file on a file unit given the number `noutbo` on opening, which becomes the default unit for writing out the object in `bigobj_write`. There are also provided separate skeleton routines intended to provide output in a format suitable respectively for visualisation by GNUPLOT and PARAVIEW, and of course routines to close files and delete the object. The precise list of member functions for bigobj_m is as follows where those also found in smallobj_m are enclosed in parenthesis:

- (initfile), open input file
- (readcon), read data from input file
- solve, generic subroutine to manage instantiation of object
- userdefined, user-defined method or function
- fn, general external function call

- dia, write object diagnostics to log file
- initwrite, open new file, making up name which defaults to having .out suffix
- (write), write out object to standard output, or to file opened by `bigobj`
- writeg, write out object suitable for visualisation by GNUPLOT
- writev, write out object suitable for visualisation by ParaView
- (delete), delete object
- (close), close input file
- closewrite, close write file opened by `initwrite`

Thus the skeleton object is defined by a formula plus data input from disc, and since both are saved internally as features, the instantiation may be deferred as necessary, so-called 'lazy initialisation'. It will be noticed that other variables in bigobj_m such as unit number are hidden, ie. cannot be accessed by other modules. In fact a common addition to the default modules is a method function `getunit` that returns `ninbo`, illustrating the approved way of accessing such data.

The reason for the emphasis on input and output from and to disc (I/O) is to facilitate the construction of a test harness for an objects in the style of bigobj_m, and indeed the aggregations of such objects. The use of "attribute-value" in I/O gives flexibility to the developer, since new variables may be added without the need to modify existing input files or output processing. Output processing is further discussed in Section **??**.

Processing Module Output

The number of I/O subroutines can be criticised. For many testing purposes an interactive debugger is adequate, and for most others that one output file is sufficient provided it is in a attribute-value format such as JavaScript Object Notation (JSON) or in a self-documenting format such as the Network Common Data Form (netCDF), to be interpreted by any suitable visualisation software.

The problem with scientific software, worse at Exascale, is the volume of data to be handled so that the ability to visualise large arrays as well as large numbers of small arrays is essential (no debugger as far as is known has such a visualisation feature). Moreover, the actionable aspect of NEPTUNE further means that any postprocessing of a generic file must also be carefully performed. Thus while a generic output file could be processed by say Linux `awk` script into a form suitable for GNUPLOT processing, this gives rise to a need for providing documentation and provenance for the script, which is at minimum a nuisance. Worse is the risk that the amount of data to be processed may be so large as

to lead to significant delay and maybe even system or other issues not handled well if at all by the script, all of which may be extremely time-consuming to resolve. Other conversion software may not be available or properly implemented on the target machine. Thus even for debugging purposes, it seems desirable that as much as possible of the processing is done within the main code, and for production runs, output in a format directly readable by say ParaView should also speed the post-processing. Since netCDF may be read directly by ParaView, this is recommended.

Code licence and availability

Code should be made available to collaborators at the earliest opportunity, to maintain close alignment between groups.

- To minimise friction and unnecessary legal restrictions on combining code components, a common, MIT licence has been adopted. This may be regarded as equivalent to the BSD3 licence in the NEPTUNE Charter, see Section **??**.
- Code development should be carried out in public repositories. The benefits of minimising delay to code use, feedback and peer review, outweigh any potential for embarrassment or code misuse.

Code style

There are many different code styles, each of which have their proponents, and can be debated at length. While everyone has their own favourite style, it seems likely that the choice of style makes little difference to objective quality or productivity. Anecdotal experience and experience from the gaming world in developing large, complicated packages (eg. Gregory **gregory**), indicate however that it is very important that there is a well-defined code style and that developers stick to it, since a mixture of styles in a code base adds unnecessary mental load and overhead. The ultimate choice is down to the project Lead, under advice from major code contributors, leading websites and textbooks, eg. in the case of the C++ language, the C++ core guidelines **cppguidelines** and more compactly Stroustrop's "Tour of of C++" **stroustroptour**.

The following style choices have been made and efforts will be made to enforce them in NEPTUNE repositories.

1. Formatting of C++ and Python will follow the style used by Nektar++
    - Developer tools: Code formatting tools should be used to automatically format code. For C++ `clang-tidy` **clang-tidywebsite** should be used while, and for Python `Black` **blackwebsite** is recommended. Similar tools should be chosen for any other languages adopted by the project.

- Enforcement: Tests run on pull requests and code pushes to the shared repository should include code formatting: The automated formatter is applied to the code, and if the output is different from the input then the code is incorrectly formatted, and the test fails.
- Documentation: LaTeX or Markdown should be used and conventions enforced regarding a maximum line length of 120 characters, restriction to ASCII character set, abbreviations, hyphenation, capitalisation, minimal use of 'z', and use of fonts to denote code names. Usage of LaTeX2HTML implies a restricted set of packages.

2. Naming

   Naming of code components (modules, classes, functions, variables etc.) is less easy to enforce automatically than formatting, but is arguably essential for NEPTUNE, because of the complexity of the physical processes to be modelled. Widely applicable good practices to be adopted are as follows:

   - Consistency: Whatever convention is used, stick to it.
   - Be descriptive: Names should be meaningful, not cryptic, and need not be very short, although brevity consistent with frequency of usage is recommended.
     - Variable names, used only on input, might consist of $3-4$ words primarily describing the variable, using 'pot-hole' convention
     - Loop or indexing variables, might consist of single characters, eg. `i`, `j`, `k`
     - Mathematical symbols, eg. those defined in Section **??**, should be converted from LaTeX into variable names using the conventions of Section **??**.
     - Where mathematical symbols are converted for use in larger expressions, the mathematical expression should be in the documentation within or linked to the code.
   - Generally prefer nouns for variables, and verbs for functions

   Given a need to mix with Object Fortran, which is not case-sensitive, the 'pot-hole' convention for naming C++ variables, ie. separating name elements by underscores, is recommended.

   Occasionally a convention is used where the name includes a part which indicates the type of the variable. For example, the JSF style for C++ **pittwhitely** recommends that pointer names begin 'p_' and that private or protected ('member') variables should have names beginning with 'm'. In general naming conventions are probably not essential, since the type can be read in the code, and modern IDEs will easily provide this information to the developer. Nonetheless, there is no objection to employing a convention,

and NEPTUNE recommend the following one for coders who wish to do so.

For C++, based on the recommendations of the book "Professional C++" (eg. ref **solterkleper**, the following prefix strings should be employed: 'm_' for member (particularly useful for indicating scope), 'p_' for pointer, 's_' for static, 'k_' for constant, 'f_' for flag (Boolean value), 'd_' for buffers/pointers on an accelerator device, and aggregations thereof, eg. 'ms_variable'. Use of global variables is deprecated, so the 'g_' prefix should not be used.

For Object Fortran naming conventions, Arter et al **fprog** codifies best practice. It may also be useful to reserve the single letters 'i', 'j', 'k' etc. for the names of loop-count variables. Loop counting should start at unity, consistent with everyday practice, unless there are good reasons for starting at zero or using offset values.

Programming languages

A small set of "approved" languages is to be used in the project consistent with the rule of 'two' described in Section **??**. This rule covers the high performance code itself and also the input/output, testing scripts and other infrastructure included in the code repository.

Important factors in the choice made included:

1. Widespread use. It must be possible for several project members at any one time to understand the language, and be able to maintain the code.
2. Stability. The code developed will potentially have a long life-span, and there are insufficient resources to continually update code to respond to upstream changes.
3. Previous usage in HPC and scientific computing. There should be an existing ecosystem of code packages, tutorials, and potential users.

The above considerations implied that the following options were extensively discussed.

- C++14, Fortran (eg. 2008), C and Python all satisfy the above criteria. For configuration, CMake, Autotools and Bash also qualify.
- SYCL (building on C++) and Julia are both less widely adopted so far, but both appear to be heading towards satisfying the above criteria and might be considered.

The recommended languages are

- As higher level DSL : Python and Julia

- For lower level HPC compatibility/DSL: Kokkos and SYCL
- General scientific work : the latest versions of C++ and (Object) Fortran, provided they are compatible with pre-existing packages and reliable compilers are available (eg. as of mid-2021 usage of SYCL implies a need for C++17.)
- For code compilation and linking etc. : CMake

Other languages may have technical merits in particular areas, or are being adopted outside scientific computing but not to a significant degree within the community. Use of these languages should be limited to isolated experiments, rather than core components. If shown to be useful in these experiments, to a level which is worth the additional overhead and risk of maintaining it, then the Lead should consider expanding the list of approved languages, consistent with the 'rule of two'.

4.3 Generating Names for Variables

The success of the project depends on an ability to code up vast numbers of complicated mathematical expressions containing a wide range of mathematical variables or symbols, illustrated by over 250 entries in Annex B to **pappeqs3**. Coding must be done as automatically, reliably and unambiguously as possible, starting with source expressions that are presumed to be in the LaTeX markup language. Unfortunately, the character set for names of C++ variables is restricted to letters of upper and lower case plus underscore '_'. Underscore will be reserved to separate words, ie. to enable use of 'pothole' convention. There is thus a challenge as to how to represent the names of variables as set out using LaTeX conventions, which involve heavy use of the escape character backslash. To enable conversion, to produce C++ equivalents, reserve 'o' as the escape character, demanding that no mathematical variable is allowed to use the letter, or its Greek equivalent omicron, even as a suffix or superfix.

A list in alphabetical order of the conventions as two character codes is provided by Table **??**.

Two character variables

Each keyboard character will be represented by one or two lowercase letters, normally those which form the first two letters of its name, ommitting 'o', thus :

- 'a' will represent 'a'
- 'aa' will represent 'A'
- 'as' will represent '*' (for asterisk)

- ‘bl’ will represent bracket on left [
- ‘br’ will represent bracket on right]
- ‘pl’ will represent parenthesis on left $\{$
- ‘pr’ will represent parenthesis on right $\}$
- ‘ti’ will represent tilde
- ‘ci’ will represent circumflex
- ‘sq’ will represent single quote
- ‘dq’ will represent double quote
- ‘st’ will represent stop or dot
- ‘ds’ will represent double stop or double dot
- ‘pu’ will represent ‘$+$’
- ‘mn’ will represent ‘-’
- ‘gt’ will represent $>$
- ‘lt’ will represent $<$
- ‘vb’ will represent $|$

Should it be necessary to use ‘o’, then ‘ooo’ will represent lowercase ‘o’ and ‘oooo’ will represent ‘O’.

Greek lowercase letters and other special characters will similarly be represented by two lowercase letters, apart from ‘o’, thus:

- ‘al’ will represent α
- ‘be’ will represent β
- ‘me’ will represent ω
- ‘ar’ will represent arrow to right $\rightarrow$
- ‘pa’ will represent parallel $\|$
- ‘pe’ will represent perpendicular $\perp$
- ‘un’ will represent underscore _

Variant Greek letters will begin with ‘v’, followed by the first letter of their transliteration into Roman letters, and uppercase Greek letters will be represented by the first and third letters, except for Π, Φ and Ψ where this is not possible, thus:

- 've' will represent ε
- 'dl' will represent Δ
- 'mg' will represent Ω
- 'py' will represent Π
- 'pf' will represent Φ
- 'pj' will represent Ψ

If storage of an expression is required, the following will be useful, thus:

- 'pt' denotes ∂
- 'ml' denotes multiplication
- 'dv' denotes division

A single letter followed by a digit will have that as a suffix, thus:

- 'n1' will denote n_1

Escape sequences

Use of different fonts will be denoted by 'o' followed by one or two digits, preceding the above codes, thus :

- 'o1' will denote uppercase, for use in conjunction with Greek alphabet
- 'o2' will denote bold(math)
- 'o3' will denote calligraphic, (math)cal
- 'o4' will denote sans-serif, (math)sf
- 'o5' will denote typewriter, (math)tt

Some of the higher integer values might be used to denote members of the same 'namespace', cf. the way sf is used to denote neutral quantities.

Positioning of suffixes and prefixes will be denoted by 'o' followed by a single letter, thus:

- 'os' indicates underneath, S for South
- 'on' denotes above, N for North
- 'oe' denotes suffix, E for East

- ‘ow’ denotes prefix, W for West
- ‘or’ denotes superfix, R for Right
- ‘ol’ denotes preceding superfix, L for Left

Other LaTeX commands will simply see their leading backslash replaced by ‘o’, thus:

- ‘onabla’ indicates ∇
- ‘otimes’ indicates $\times$

Chapter 5

Design Justification File

The Design Justification File (DJF) is generated and reviewed at all stages of the development and review processes. It contains the documents that describe the trade-offs, design choice justifications, verification plan, validation plan, validation testing specification, test procedures, test results, evaluations and any other documentation called for to justify the design of the software. A wide range of ideas for NEPTUNE has been considered both by UKAEA and NEPTUNE grantees, as indicated by the titles in Table **??**, and choices described in the DDF will be found supported by by the minutes of meetings which form part of many reports. To see what has been considered before, it is recommended that the access-controlled github repository **xpndocswebsite** be cloned to a local directory and its subdirectories `reports` and `reports/ukaea_reports` be indexed using software such as DocFetcher **docfetcherwebsite** or Recoll **recollwebsite**.

5.1 UKAEA (Internal) Reports

Table 5.1: **NEPTUNE REPORT TITLES** For full filenames, prepend the string 'CD-EXCALIBUR-FMS'. Source is in subdirectory 'tex'.

No.	Key bibtex	Title	Filename PDF	Source .tex file
0001	**sciplan**	ExCALIBUR Fusion Modelling System Science Plan	0001	sciplan/
0004	**y12acts**	ExCALIBUR Fusion Modelling System Activities Y1-2	0004	-
0010	**y1re111b**	NEPTUNE : Report on Y1 2020 External Workshop (REPORT1)	0010-M1.1.1b	-
0011	**y1re121**	Year One Summary Report	0011-1.00-M1.2.1	t12/rp1
0012	**y1re211**	Options for Geometry Representation	0012-1.00-M2.1.1	t21/rp1

0013	**y1re231**	Options for Particle Algorithms	0013-1.01-M2.3.1	t23/rp1
0014	**y1re311**	NEPTUNE : Report on system requirements	0014-1.00-M3.1.1	-
0015	**y1re331**	NEPTUNE : Background information and user requirements for design patterns	0015-1.00-M3.3.1	-
0016	**y1re351**	Benchmarking requirements for NEPTUNE and available tools	0016-1.00-M3.5.1	-
0018	**y1re111a**	NEPTUNE : Report on Y1 2019 Internal Workshop	0018-M1.1.1a	-
0020	**charter** EX-CALIBUR NEP-TUNE Charter	0020	chart/rp	
0021	**pappeqs3** Equa-tions for EX-CALIBUR Proxyapps	0021-1.20-M1.2.1 /NEPTUNE	rp21/rpc	
0022	**y2re312**	Report on user frameworks for tokamak multiphysics	0022-M3.1.2	t31/rp2
0023	**y2re332**	Report on design patterns specifications and prototypes	0023-M3.3.2	t33/rp2
0024	**y2re313**	Report on user layer design for Uncertainty Quantification	0024-M3.1.3	t31/rp3
0025	**y2grant**	ExCALIBUR Fusion Modelling use case: contract award recommendation report	0025-M1.3.1	docx
0026	**y2re333**	Design patterns evaluation report	0026-M3.3.3	t33/rp3
0027	**y3act**	ExCALIBUR Fusion Model SPF Research Plan Y3	0027-M1.5.1	y3act/g
0030	**y2re141**	Winter 2020-21 Workshop	0030-M1.4.1	t14/rp1
0030a	**y2close**	ExCALIBUR NEPTUNE Project analysis to date: close out Y2	0030a-M1.6.1	docx
0031	**y2re251**	Select techniques for MOR (Model Order Reduction)	0031-M2.5.1	t25/rp1
0032	**y2d33**	Module Guide	0032-D3.3	d33/rpc
0033	**y2d34**	Development Plan	0033-D3.4	d34/rpc
0034	**y2re221**	Performance of spectral-hp element methods for the referent plasma models	0034-M2.2.1	t22/rp1
0035	**y2re241**	Assessment of which UQ methods are required to make NEP-TUNE software actionable	0035-M2.4.1	t24/rp1
0036	**y2re271**	Identification of suitable preconditioner techniques	0036-M2.7.1	t27/rp1
0037	**y2re281**	Selection of the physics models	0037-M2.8.1	t28/rp1
0038	**y2re261**	Identification of a preferred overall numerical scheme	0038-M2.6.1	t26/rp1
0039	**y3re321**	Survey of code generators and their suitability for NEPTUNE	0039-M3.2.1	t32/rp1
0040	**y3re51**	Management of external research. Supports UQ Procurement	0040-M5.1	t51/rpc

0041	**y3re322**	Survey of Domain Specific Languages	0041-M3.2.2	t32/rp2
0042	**y3re314**	Specification and Integration of Scientific Software	0042-M3.1.4	t31/rp4
0043	**y3re242**	Selection of techniques for Uncertainty Quantification	0043-M2.4.2	t24/rp2
0044	**y3re252**	Selection of techniques for Model Order Reduction	0044-M2.5.2	t25/rp2
0045	**y3re272**	Identification of suitable preconditioner techniques	0045-M2.7.2	t27/rp2
0046	**y3re212**	Surface mesh generation	0046-M2.1.2	t21/rp2
0047	**y3re222**	Finite Element Models: Performance	0047-M2.2.2	t22/rp2
0048	**y3re232**	Options for Particle Algorithms	0048-M2.3.2	t23/rp2
0049	**y3d32**	Domain-Specific Language (DSL) and Performance Portability Assessment	0049-D3.2	d32/rp
0050	**y3d35**	Verification and Benchmarks Methodology	0050-D3.5	d35/rp
0051	**y3re61**	Finite Element Models: Complementary Activities I	0051-M6.1	t61/rp
0052	**y3re71**	Literature review for Call T/AW086/21: Mathematical Support for Software Implementation	0052-M7.1	t71/rep
0053	**y3re72**	Code coupling and benchmarking	0053-M7.2	t72/rp
0054	**y2d31**	Software Specification Web-site	0054-D3.1	d31/rp
0055	**y3re181**	Report of NEPTUNE Workshop 7 October 2021	0055-M1.8.1	t18/rp1
0056	**y3grant**	ExCALIBUR Fusion Modelling use case: contract award recommendation report	0056-M1.7.1	docx
0057	**y3re262**	Fluid Referent Models	0057-M2.6.2	t26/rp2
0058	**y3re282**	Technical report on Physics model selection	0058-M2.8.2	t28/rp2
0059	**y45act**	ExCALIBUR -Fusion Model System Y4-Y6	0059-M1.9.1	y45act/
0060	**y3close**	Analysis to Date: Close out Y3	0060-M1.10	t110/rp
0061	**y3re42**	2-D Model of Neutral Gas and Impurities	0061-M4.2	t42/rp
0062	**y3re43**	High-dimensional Models Complementary Actions 2	0062-M4.3	t43/rp
0063	**y3re52**	Selection of techniques for Uncertainty Quantification	0063-M5.2	t52/rp
0064	**y3re62**	Finite Element Models Complementary Actions 2	0064-M6.2	t62/rp
0065	**y3re73**	Software Support Complementary Actions 2	0065-M7.3	t73/rp
0066	**y3re41**	Support High-dimensional Procurement	0066-M4.1	t41/rp

Brief Survey of Reports to end FY 2020/21

For Activity 2, UKAEA produced three milestone reports, namely

- CD/EXCALIBUR-FMS/0012-M2.1.1 - Options for Geometry Representation
- CD/EXCALIBUR-FMS/0013-M2.3.1 - Options for Particle Algorithms
- CD/EXCALIBUR-FMS/0031-M2.5.1 - Select techniques for MOR (Model Order Reduction)

together with the Activity 1 Report ('Equations document')
CD/EXCALIBUR-FMS/0021-1.00-M1.2.1 - Equations for NEPTUNE/ExCALIBUR proxyapps

The first two (Reports 12 and 13) describe the problems presented by the fusion use case in respect of geometry and strong magnetic field in the first, and of generally but not invariably low collisionality of the edge plasma in the second. They set out issues that needed to be urgently addressed and possible lines of research. Report 31 discusses in depth a wide range of options for research into Model Order Reduction, drawing attention to the possibility of producing scalable algorithms by borrowing ideas from the field of Data Assimilation. Report 21 sets out equations to be studied using the first six proxyapps, beginning with relatively simple models for anisotropic transport and advancing to complex models of plasma-neutral interaction.

For Activity 3, UKAEA produced six milestone reports, namely

- CD/EXCALIBUR-FMS/0022-M3.1.2 - User frameworks for tokamak multiphysics
- CD/EXCALIBUR-FMS/0024-M3.1.3 - User layer design for uncertainty quantification
- CD/EXCALIBUR-FMS/0023-M3.3.2 - Design patterns specifications and prototypes
- CD/EXCALIBUR-FMS/0026-M3.3.3 - Design patterns evaluation
- CD/EXCALIBUR-FMS/0032-D3.3 - Module Guide
- CD/EXCALIBUR-FMS/0033-D3.4 - Development Plan

and there were three earlier milestone reports from FY2019/20, namely

- CD/EXCALIBUR-FMS/0014-M3.1.1 - NEPTUNE: Report on system requirements
- CD/EXCALIBUR-FMS/0015-M3.3.1 - NEPTUNE: Background information and user requirements for design patterns
- CD/EXCALIBUR-FMS/0016-M3.5.1 - Benchmarking requirements for NEPTUNE and available tools

These reports are concerned principally with assessing the state-of-the-art in software design with a particular emphasis on design of scientific software, and of course paying attention to Exascale applicability. Selected textbooks and the wider literature were examined for the factors important for successful software developments. This examination threw up the importance of (1) software frameworks described in Report 22 as an integrated set of software artefacts that collaborate to provide a reusable architecture for a family of related applications, (2) software layering in Report 24 as a more widely useful technique than just one enabling 'separation of concerns', and (3) design patterns in Reports 15, 23 and 26 as an approach to reusing and communicating

reliable software structures. The importance of building a community and the techniques for doing so were also described in Report 22.

Report 32 describes how the concept of module or class should be integrated into a structure of frameworks, layers and design patterns, so that a large, complex code can be partitioned into manageable segments. The utility of the Unified Modeling Language (UML 2) to describe not just software structure, but also code use in a wider based engineering structure, through model-based systems engineering (MBSE) was noted. Report 33 attempts a preliminary synthesis of the material presented in the previous Activity 3 reports into a plan for the $\mathrm{NEPTUNE}$ software life-cycle, with focus on the subdivision of the plan into documents expected to be arranged as a web-site. Report 24 also includes an introduction to a wide range of uncertainty quantification (UQ) techniques.

Brief Survey of Reports FY 2021/22

For Deliverable 4, UKAEA produced three milestone reports, namely

- CD/EXCALIBUR-FMS/0066-M4.1 Support High-dimensional Procurement
- CD/EXCALIBUR-FMS/0061-M4.2 2-D Model of Neutral Gas and Impurities
- CD/EXCALIBUR-FMS/0062-M4.3 High-dimensional Models Complementary Actions 2

The first (Report 66) supports the usefulness of the call for development of higher dimensional elements, suitable for solution of a continuum kinetic model of plasma, by demonstrating Nektar++ solution of a $1d1v$ model. The second (Report 61) considers the use of the Julia language as a means to coding particle models of plasma and neutral gas at HPC, with broadly favourable conclusions. The third and final (Report 62) outlines critical physics expected to require treatment using particle-based and/or Monte-Carlo methods. Its principal content examines how best to treat inter-processor communication of particle-based information at the Exascale. There is also a brief description of a proxyapp for the exploration of $1d1v$ solution by particles.

For Deliverable 5, UKAEA produced two milestone reports, namely

- CD/EXCALIBUR-FMS/0040-M5.1 Management of external research. Supports UQ Procurement
- CD/EXCALIBUR-FMS/0063-M5.2 Selection of techniques for Uncertainty Quantification

The first (Report 40) begins with a reminder of the high level of “noise" in tokamak data, and the kind of comparisons with simulation required. It is pointed out that although spline interpolants are provably optimal, they may perform poorly for classes of functions relevant to the tokamak edge. The main

content is the use of the VECMA toolkit for UQ of BOUT++ and Nektar++ for two 2-D fluid dynamical models. There is also derivation of simplified model by dimension reduction, by integration in one coordinate and/or by use of the Lie derivative. The report finishes with a summary of UQ-related PhD projects sponsored by NEPTUNE.

The second (Report 63) pursues the use of splines for NEPTUNE, indicating utility in the case of noisy data, and explores the use of Gaussian processes, including derivation of key formulae, and highlighting their strengths relative to splines. There is a brief comparison with neural network surrogates in an annex.

For Deliverable 6, UKAEA produced two milestone reports, namely

- CD/EXCALIBUR-FMS/0051-M6.1 Finite Element Models: Complementary Activities I
- CD/EXCALIBUR-FMS/0064-M6.2 Finite Element Models Complementary Actions 2

The first (Report 51) details the application of the Nektar++ spectral / hp finite-element software to the classic problem of two-dimensional vertical natural convection - physically a model for the heat transfer that takes place within the cavity of a double-glazing unit and also relevant to heat transport in a plasma. A brief survey of results from the literature is followed by a numerical investigation showing transitions between conducting, laminar convective, and turbulent regimes. Small extensions to the existing Nektar++ framework, aimed at extracting engineering-relevant quantities such as the maximum near-wall temperature, are given. The second (Report 64) builds on these results with a quantitative comparison to the well-established MIT numerical convection benchmark and also the reproduction of some detailed flow-fields from a recent publication; excellent agreement between results from Nektar++ and those from the literature is obtained in both cases. Also included in this report is a preliminary study of a numerical implementation of discrete exterior calculus with a demonstration of spectral convergence for a simple test problem, work motivated by the favourable properties of such schemes when coupled to particle kinetic codes.

For Deliverable 7, UKAEA produced three milestone reports, namely

- CD/EXCALIBUR-FMS/0052-M7.1 Literature review for Call T/AW086/21: Mathematical Support for Software Implementation
- CD/EXCALIBUR-FMS/0053-M7.2 Code coupling and benchmarking
- CD/EXCALIBUR-FMS/0065-M7.3 Software Support Complementary Actions 2

The first of these (Report 52) is a literature review performed to support the Call T/AW086/21: Mathematical Support for Software Implementation. It describes

recent advances in algorithm development for hyperbolic and elliptic equations. In particular, the report describes developments in IMEX schemes, Deferred Correction methods, Asymptotic Preserving (AP) methods, and Variable Stepsize, Variable Order (VSVO) timestepping for hyperbolic problems, and AP methods and nested solvers for elliptic problems.

The second (Report 53) provides a commentary on the present state of Exascale hardware and software, and discusses the available tools and technologies available for benchmarking and code coupling. The hardware and software landscape of HPC systems is becoming increasing diverse, with a proliferation of vendors and different technologies. To perform well at Exascale, software will likely need be able to target multiple heterogeneous systems. In such an environment, it is crucial for developers to have access to benchmarking infrastructure to measure performance and highlight regressions. This environment and the separation of concerns approach taken to navigate it necessitates the development of discrete proxyapps which will need to be drawn together into a single software suite. Thus the issue of code coupling will also be important at Exascale.

The final (Report 65) describes aspects of coordination within the NEPTUNE project not covered in previous reports, namely, the development of a GitHub repository for infrastructure code and project planning; the development of a project website for knowledge transfer within NEPTUNE ; and a description of collaborations arising from NEPTUNE -related interactions, including the Fusion Modelling Use Case working group established create a connection between Project NEPTUNE and the wider ExCALIBUR programme.

Parallelism Abstraction

To exploit parallelism most effectively on any given architecture, data must be arranged in arrays to which the same operations can be applied to many (N_{adj}) adjacent elements. The arrangement of data describing, say, the magnetic field or a particle distribution function can nonetheless make a big difference to ultimate speed of execution which can depend sensitively on N_{adj}. Thus a good API could be defined at the array level, taking away from the developer the decision as to whether the data is arranged as $n_x \times n_y \times n_z$ or $n_z \times n_x \times n_y$, ie. as to which array index runs the fastest. Further, extremely large first indices n_x might for example be factored so that the first index is of order 64 to exploit caching, whereas the final index might be used to map array contents to different nodes of the machine.

```
Address of array
General indexing (start at 0)
I(0)*INC(0)+I(1)*INC(1)+I(2)*INC(2)+I(3)*INC(3)+...
Suppose I index from 0 to N0-1
J index from 0 to N1-1
K index from 0 to N2-1
```

```
L index from 0 to N3-1
Address :
(I,J,K,L,...), I=0,N(0), J=0,N(1), K=0,N(2), L=0,N(3)
Set INC(0)=1, INC(1)=N0, INC(2)=N0*N1, INC(3)=N0*N1*N2
N(0)=N0-1, N(1)=N1-1, N(2)=N2-1, N(3)=N3-1
Address :
(L,I,J,K,...) Set INC(0)=1, INC(1)=N3, INC(2)=N0*N3, INC(3)=N0*N1*N3
N(0)=N3-1, N(1)=N0-1, N(2)=N1-1, N(3)=N2-1
(0123)->(3012) is permutation
integer value 1, labels 3, 30, 301,.. give increments
Set N(i)=1 to suppress dependence on index i
```

In a physics modelling code, it seems reasonable that physics should have a say as to how the data is arranged, with the special implication that all information relating to a particular position in space should be as close together as possible. However, particularly for edge physics, this may lead to conflict with an array level API. There are two main problems, namely that at a given spatial point (1) some species may be represented as particles and others as finite elements and some as both, and (2) not all species need be present at a given point. The issue at (1) arises when the species collisionality varies so that a fluid and a high-dimensional representation that accounts more accurately for non-Maxwellian effects are needed in different spatial regions, with the different representations allowed to overlap. Situation (2) may occur with a neutral species that becomes fully ionised with distance into the plasma, or when say singly-charged ions of a certain species are only present in the divertor. The problem is intensified when p-adaptive finite elements are used such that adjacent elements may have different orders of polynomial discretisation. It may also be desirable when working with ensembles to have samples from different solutions but for the same spatial region to be physically close together in storage.

The plasma physical constraint may be met by domain decomposition in position space, so that within each subdomain, fluid species can be represented by one set of arrays, one per species, and particles or other high-dimensional representations as other set(s) of arrays. The optimality of this arrangement, and certainly the size of subdomain, depends on machine architecture. For example, on a node with both conventional CPU cores and a GPU, it might be good to store finite elements adjacent to the CPU and use the GPU for particles. Another option might be to take the localisation concept to its extreme, and arrange together quantities that are close in the 6-D phase velocity and position space, perhaps using an hierarchy of elements in velocity space. Fluid species might be represented by pointers in these elements, without too much wastage of store, even if there is only one species that requires a high-dimensional representation.

Since the main work of a NEPTUNE solver is expected to be the numer-

ical inversion of a large matrix to obtain field values at a new time or iteration, there is even a question mark over how much weight should be attached to the localisation constraint. At the Exascale, the matrix and especially its preconditioner must be virtual in the sense that it will be too large to store all the coefficients simultaneously, given the size of field discretisation. Hence the ease of computation of the coefficients of the matrix may be more important for performance.

Chapter 6

Design Definition File

The Design Definition File (DDF) is a developer-generated file that documents the result of the design engineering processes.

6.1 Design specification

Template given. Describe modules and hierarchy of modules. Also describes likely and unlikely changes to the code. Anticipated changes guide the design: ideally a change affects only one module

The design specification may need to be supplemented by a Module Interface Specification (MIS). More concrete in defining access routines and syntax, but still abstract in not defining how things are done.

Use of VECMA/SEAVEA as framework UQ by ensembles, active subspaces (from **y3re242**) and GPs for surrogates (from **y3re252**).

The proposed NEPTUNE development is of sufficient complexity that the production of code should be as automatic as as possible, and the 'write once, use many times' principle implies that the starting point for code generation will often be LaTeX or Markdown format documents from the DJF or indeed DDF.

For NEPTUNE, the initial write should be in LaTeX format representing an extension of the tabular layout used to generate Table **??**, for conversion to DOXYGEN input format for documentation and C++ source, specifying:

- variable name, which for NEPTUNE should be generated from LaTeX using substitutions set out in Table **??**.
- brief description, to remind user what is the purpose of the variable
- units
  - physical units should be SI, except eV for temperatures and mm for CAD inputs

  - scale factors for extreme-valued fields, eg. 10^{18} for number densities, or for quantised fields, eg. position expressed in units of separation of a uniform grid.

- default value(s) on input
- simple constraints on variable values
  - whether real number or integer-valued
  - range specification, eg. $0 < n \leq 10$ if n must be a positive, small integer
- detailed description of what variable does, if not covered by group description.
- constraints in terms of other variables

Input variables are grouped according to the objects/classes which they help define.

SMARDDA-MISC illustrates how to produce software that auto-generates the equivalent of .h files to describe objects and and .cpp or .m files for

1. setting default values of variables
2. dumping inputs to .log text-files if required
3. checking constraints on input variables
4. saving acceptable values

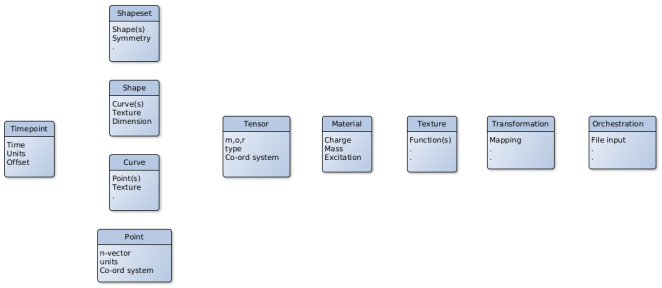


Figure 6.1: NEPTUNE base classes.

6.2 Objects/classes

Notes Procedures denoted in boldface, as separate sections.
n-D does not include time
Refs **sciplan** in exc.bib

Base classes

The proposed base classes for NEPTUNE are listed below and shown graphically in Figure **??**.

- Timepoint : point in physical time Attributes of time, units, offset (Alternatively just real scalar)
- Point : point in *n-D* space
  many make curve, shape;
  is particle location in *n-D* ; Attributes of n-vector, units, coordinate system (Alternatively just real n-vector)
- Curve : parts are one or more points, straight lines or textures or from CAD input or CSG input;
  many make shape;
  is shape boundary, is particle trajectory, is ray
- Shape : parts are curves and textures, planar rectangles, or surfaces from CAD input or CSG input;
  many make Shapeset;
  is surface which aggregates as BC
- Shapeset : parts are shapes, regular lattice, or volumes from CAD input or CSG input;
  is finite element geometry, is unstructured mesh, is surface geometry of body, is volume in *n-D*, $n \geq 3$;
  helps defines field
  Attributes of degree of toroidal symmetry

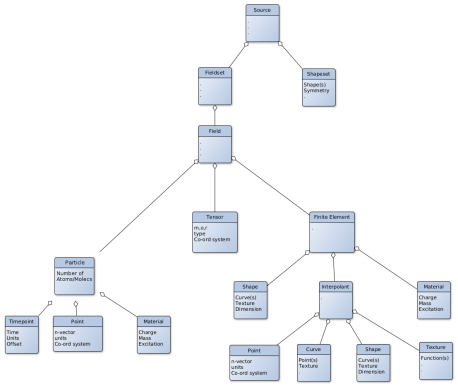


Figure 6.2: Aggregation of base classes to form a class 'Source'.

- Tensor : parts are m numbers at a point, order o, type eg. *udd*, and density r in *n-D* coordinate system of type *c*;
  is $(m = 3,\ o = 1,\ n = 3)$ velocity, is $(m = 1,\ o = 3,\ n = 3)$ density,
  is $(m = 1,\ o = 0,\ n = 3)$ temperature, is $(m = 3,\ o = 0,\ n = 0)$ is array;
  many help make field (*u* denotes contravariant, *d* denotes covariant, *c* defines cartesian, cylindrical, toroidal coordinates, $r = 0$ usually)

- Material : from database input ;
  helps make body, particle, many make matexture, plasma
  Attributes of charge, excitation level and mass

- Texture : parts are mathematical library functions, particularly mathematical library interpolation functions - see Section **??**
  aggregates as matexture, BC

- Transformation : mathematical formula defining geometry transformations on point and tensor (co- and contra-variant) $\bar{\mathbf{x}} \rightarrow \mathbf{x}$

- Orchestration : parts are from configuration file input see **Orchestration**, model, framework

Aggregates

- Particle : parts are location, velocity, material;
  Attributes of particle weight

- Interpolant : parts are points, curves, shapes, textures, or timepoints, textures

- Diagnostic : parts are DSL input instructions **Diagnostic Processing**, fieldset

- FE (Finite element) : parts are shape, interpolant, material
- Field : parts are tensors and finite elements, or particles; many make fieldset
- Fieldset : parts are fields
- DE (Differential equation) : parts are operators (DSL input), IC, BC and source
- Model : parts are **Solution of DEs**
- Source : parts are shapeset, fieldset. See Figure **??**.
- Matexture : parts are materials, textures
- Body : parts are shapeset, matexture
- BC (Boundary Condition) : parts are surface, material, texture
- GEOQ (Geometry plus B-Equil) : parts are shapeset, field

Simple inherits

- HDS (Hierarchical Data Structure) : multi-octree, is a shapeset
- Trajectory : particle position as time varies, is a curve
- IC (Initial Condition) : is a fieldset

Solution of Differential Equations

Use in part ABSTRACT CALCULUS and PUPPETEER patterns (cf. GoF FACADE) from Rouson et al. **rousonxiaxu**, see Section **??**.

Diagnostic Processing

1. Read configuration file
2. Determine whether any diagnostic needed at present physical time
3. Select input fieldset
4. Select diagnostic type, use in part ABSTRACT CALCULUS from Rouson et al. **rousonxiaxu**
    - Initial logs
        - UUID
        - Key input data

    - Key properties, eg. LCFS
  - Combinations of
    - Field / quadratic field (eg. power, flux quantity) / general formula
    - Point, line integral, surface integral, volume integral
  - Mass/charge, momentum/current and power balances
  - Turbulence statistics - cross-correlations, spectra (not particle, ray)
  - Difference between solutions/experiment (RMS as 'skill') (not particle, ray)
  - See "emergent physics as diagnostic" in imas_objects.tex
5. Calculate output fieldset
6. Set output format
7. Output fieldset to disk, screen

Orchestration

1. UQ Framework VECMAtk and FabNEPTUNE
2. GUI
3. CLI
4. Possible restart (OLYMPUS logic, Fig.1 of **y2re312**)
5. **Initialise** from functions.md
6. **Solution** from functions.md

6.3 Execution sequence

Preprocessing and components

- Convert equilibrium B-field to standard format (ie. a standard resembling EQDSK)
- Convert n-D geometry (NURBS) to opensource format

Initialise

- Convert n-D mesh to readable format
- Assign boundary conditions to mesh (finite element by finite element)
- Validate inputs (eg. Python Cerberus)
- Initialise fields, possibly exploiting toroidal symmetry
  - using sources, calculate with possibly simplified dynamics
  - semi-analytically (Gaussian in vel. space)
  - random perturbations
  - from disk file
  - from previous calculation
  - apply operators to field (eg. EQDSK data $\rightarrow$ B-Equil)
  - combine vacuum field and B-Equil
- Calculate LCFS
- Normalise field
- Distribute mesh across processors
- Distribute particles across processors

Solution

- Extra outer loop over parameters (UQ Framework)
- Outer loop over time
- Inner loop over solvera
  - loop over particles
  - loop over rays
- More deeply nested iteration

- Convergence test
- Rouletting particles
- Particle collision
  - with geometry
  - with other particle
- Adaptive meshing
- Construct surrogates (UQ Framework)
  - Reduce in n-D ($n \to n-1$)
  - Combine particles
  - Replace particles by FE
  - Smoothing
  - Gyro-averaging
  - Sparse models for Data Assimilation (SINDy)
  - Gaussian Process
  - Polynomial Chaos Expansion
  - Active Subspaces
  - PGD
- Connect surrogates (UQ Framework)

Assortment

- Calculate HDS from Field geometry data
- Intersect triangle with cuboid
- Calculate surface normal and tangents, curvatures
- Calculate curve tangent and normals, torsion, curvature
- Calculate volumes
- Locate point in field element geometry data
- Select algorithms
- Label with physical units, array dimensions, transformations
- Increase in n-D ($n \to n+1$)
- Replace FE by particles
- Evaluate FE representation at set of points

Diagnostics

- Calculate terms in power balance

Utilities

- Format conversion
- sorting

Mathematical library operations

- FFT (FFTW package)
- linear algebra
- quadrature
- optimisation
- geometry transformations on point and tensor
- Interpolation STRATEGY to select among
  - splines (2-D, 3-D)
    - regular (de Boor)
    - local
    - Weiland
  - Fourier interpolation
  - rational interpolation
  - Lagrange interpolation (after Trefethen)

6.4 Design patterns

The design patterns likely to be most relevant to NEPTUNE are described in refs **y2re332**, **y2re333**.

The PUPPETEER pattern appears to be exclusive to the works of Rouson et al, hence is described further here. It is directed towards the calculation of the Jacobian of multi-component systems typically needed for Newton-Raphson type solution of update equations. The idea is that the puppeteer need only know which blocks of the Jacobian matrix are nonzero, although the puppets need to be capable of forming derivatives with respect to all state variables.

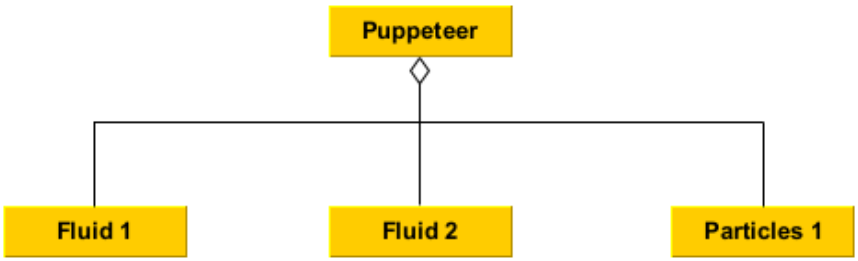


Figure 6.3: PUPPETEER pattern. Each puppet ('Fluid 1', 'Fluid 2', 'Particles 1') may include terms involving the other puppets. For example, suppose the puppet for 'Fluid 1' has terms involving also 'Particles 1' variables (but not 'Fluid 2'), then the puppeteer can request derivatives of the puppet's terms both with respect to 'Fluid 1' and 'Particles 1' variables, but knows not to ask for derivatives with respect to 'Fluid 2' variables.

Chapter 7

Management File

The Management File (MGT) is a developer generated file that describes the management features of the software project notably, organizational breakdown and responsibilities, work activities breakdown, selected life cycle, deliveries, milestones and new risks. Note that the report **y2d34** entitled "Development Plan" is primarily concerned with the documents needed for managing the project, and how they should be arranged as a website.

The range of levels of detail and implicitly computational cost (from Exascale down to laptop) will follow from the projected development via a sequence of proxyapps of increasing complexity and detail. This process is expected to enable an enhanced selection through experience and feedback, of

- a common set of software objects/classes
- computer languages suitable for design with 'separation of concerns
- numerical algorithms
- user interfaces
- interfaces to databases and coupling to codes not conforming to NEPTUNE standards

Selections will be recognised as better if they lead to improvements in scalability and portability, reliability and resilience, extensibility, and ease-of-use.

7.1 Introduction

Webpages on this site concerning management details are taken from the report **y3re314**, based largely on the work of Ben Dudson, which presents guidance concerning the mechanics of an opensource development by a community distributed across different sites and organisations, intended to produce software intended for widespread long-term usage. Parts of ref **y3re314** relate more

to the technical specification (TS) and are reproduced in Section **??**, another part concerns operational aspects (OP), see Section **??**. Its recommendations are broadly consistent with those laid out by Bungarth & Heister **Ba13What**, and in particular those for the usage of `git` conform to practice recommended by the ITER organisation. This document is not the place for a general discussion of software engineering practices, and does not cover code coupling, both of which topics are discussed in the open literature, see in particular Lawrence et al **La18Cros** for HPC software engineering and Belete et al **Be17over** for code coupling, also see other NEPTUNE reports, particularly refs **y2re312**, **y2re333**, **y3re72**.

Ref **y3re314** assumes that all the community has signed up to a "Charter" which for NEPTUNE appeared as report **charter**, reproduced in Section **??**. The guidance**y3re314** includes important issues that need to be agreed as early as possible. These include practical points concerning frequency of meetings, code review etc., designed to ensure efficient collaboration between a wide group of project partners. The guidance document**y3re314** also seeks not merely to prescribe, but to give compelling arguments for the choices made in respect of guidelines. Acknowledging the possibility of diagreements, it states that efforts will be made to ensure consensus or at least agreement between the two most affected project partners on any decisions taken. However, in the event of continuing disagreement, the technical leader or 'Lead' for the project will ultimately decide on the basis of technical evidence presented, subject to ratification by higher management.

One general rule is always to allow two options ('rule of two'), intended to enable exploitation and possible incorporation of any promising new software (eg. package, library or language) or relevant algorithm which emerges during the course of the project. Since however, each option doubles the potential cost of developing and maintaining software, a good case must be made to the Lead for a new option, and the innovator include provision for retiring one of the existing options should there already be two. Implicitly thereby, as discussed at the end of Section **??**, a third exploratory option is also allowed.

A similar recommendation (rather than rule) regarding both code and documentation is to 'write once, re-use many times'. This to a large extent explains a preference for LaTeX2HTML as enabling multiple reuse of the same text and mathematical expressions in different documents and on different webpages, via use of LaTeX input command.

7.2 Management

Meetings, whether on-line or in-person are regarded as critical for good collaboration, and are discussed in Section **??**. The other key collaborative element centres naturally on the software, where use of the `git` control system, see Section **??** and consequent use of repositories, see Section **??**, is becoming universal.

Meetings and Workshops

To start the project and any notable identifiable piece of work within it, a kick-off meeting should bring together all partners who will contribute significant code to the project. The aim of this meeting will be to build personal links among the team, and to establish community practices consistent with the charter. Efforts should be made to build consensus and a community spirit within the project team.

A regular project planning and monitoring meeting should be set up, at least monthly. Its agenda should include short updates on progress of each project component, and focus on the project planning and coordination. In addition, a separate series of seminars and training should be organised, where each partner might give a longer talk on an aspect of their work, for example showing other partners how to use recently developed capabilities.

Development and collaboration mechanisms should include:

1. A system of code repositories for version control (eg. `github`)
2. Automated testing infrastructure (eg. `github` actions)
3. Documentation infrastructure, ie. as a website
4. A repository for long-term storage of large files, records of meetings, presentations etc. (eg. Google shared drive)
5. A chat/messaging service such as Slack, to facilitate interactions between developers

As these are established, a series of training workshops should be arranged. These should include talks on the "high level" objectives, on the near-term plans of each partner, and also hands-on training in the tools being used.

Version control

The standard `git` version control system should be used; there is no viable competitor to this in terms of capabilities, widespread adoption, or integration into other tools and services (eg. `github`).

A common complaint against `git` is the user interface, which can be intimidating to new users. There are very strong reasons why even programmers

with plenty of other experience, should seek guidance and preferably training in use of the command line interface (CLI). For those who have time enough to attempt to do so without, a few hints are provided:

1. The complexity of the interface can be mitigated by restricting usage to a few well-chosen subcommands such as `clone`, `add`, `commit`, `push`, `pull`, `diff`, `log` and `status`.

2. Exercise caution before using other subcommands or new options to the core subcommands, eg. by first committing all files, adding a suboption which indicates what will be done without actually modifying any files, and avoiding forcing options.

3. For the purpose of the key subcommands such as 'pull' and 'push', it is important to remember than these are are defined from the user's point-of-view, so that 'pull' brings source from the repo towards the user, and 'push' sends it away. There are other non-intuitive aspects so that it is important to study very carefully the description of any new sub-command/option and particularly its ordering of options.

4. Since the software is widely used, error messages can invariably be 'googled' for further explanation.

5. Should conflicts occur, these are recorded by the insertion of strings '`+++...`', '`>>>>...`' and '`<<<<...`' in disc files to indicate lines where the clashes lie. Many users find resolving conflicts very difficult on the basis of such information, however making up for the absence of a GUI mechanism within `git` to do this, it is possible to integrate GUIs such as **meld**, being aware of possible system dependences.

Otherwise, the experience of `git` can be mitigated through:

- Training: Links to training material for adopted tools should be made available as part of the project documentation. This should be supplemented by training, both one-to-one and as part of a programme of talks and training.

- Adoption of, and training in, tools to provide easier interfaces. `github` itself allows browsing of history; `Magit` is an excellent interface integrated into Emacs; and similar tools exist for eg. Visual Studio Code. The ITER organisation uses `bitbucket` and UKAEA uses `gitlab`.

Code repositories

The structure of ExCALIBUR will result in a number of different components, experimental proxyapps, and increasingly complex applications. There are two main different approaches as to how these different components could be split

between git repositories, namely (1) Several large code bases are kept in a single repository (a 'monorepo') and (2) projects are kept in separate repositories, with dependencies being included as `git` submodules.

Adopted is a compromise approach whereby:

- A `github` 'organisation' `https://github.com/ExCALIBUR-NEPTUNE` was created to host new repositories. Organisations allow permissions for groups of administrators and developers to be managed, and this is currently restricted to community members, so that it is important to be 'logged in' to access their components.

- Individual components and proxyapps are hosted in separate repositories under this organisation. These contain the code, unit tests, documentation etc. specific to these components.

- Reports produced as part of the NEPTUNE project, unless they contain commercially sensitive information, are to be found in the repository **xpndocswebsite**.

- A central repository under this organisation includes components as submodules. These could be organised into a directory structure, with documentation explaining the relations or coupling between components. In this repository should go:

  - Integration tests which couple components and ensure that they work together
  - Documentation of the interfaces between components, project conventions (eg. style guides), and overall project aims.

  Sub-modules are pinned to a particular `git` commit, so that at any point the versions included are those which are known to work with each other. A developer who wants the latest version of a component should clone the individual repository, while a user who wants something that "just works" should clone the central repository.

(There is the disadvantage of a tie specifically to `github`, but loss of the 'organisation' capability would be expected to be an inconvenience rather than a disaster for a project.)

Development workflow

The standard `git` work flow has been adopted, since this is widely familiar and has been developed as best practice based on industrial experience. Exceptions are allowed for minor issues, such as typographical errors and broken links in documentation.

Each code component maintains a `main` branch (often referred to as the 'master' as in 'master copy'), which can only be modified through a pull request mechanism which ensures peer review and testing. Bug fixes and feature development are done in separate branches, either in the same repository, or in forked repositories. When someone encounters a bug, or wishes to develop a new feature, the recommended approach ise:

1. An issue is opened, describing the bug or feature request or proposal. This allows discussion of the issue, and possible approaches to addressing it.

2. A pull request is opened as early as possible, marked "Work in progress" or similar. This can contain only minimal code or outline of the code structure. This links to the issue, lets other people know that it is being worked on, and enables peer review and input into the development direction.

3. Once ready for merging, and consensus has been reached that the proposed change should be made, then it is merged.

If a code is sufficiently large, then a further degree of separation between the stable `main` branch and active development is needed. A common pattern is to only branch off and merge features into a `next` branch. Periodically this branch is merged into `main` as a new release, once the new features are judged to be sufficiently mature and tested.

Whether into `main` or `next`, pull requests should be reviewed using a checklist the remind the reviewers. Review involves testing, aspects of which are addressed in Section **??**.

It must be stressed that code review is not a job separate from code development: **All developers should be expected to participate in and carry out code reviews**. Reviewing code benefits not only the original author, but also the reviewer. Through the discussion, it contributes to a sense of shared ownership of the code base, and spreads good practices. There is the implication that code should be written 'for the other guy', ie. so that the other guy can understand it without much difficulty. It also ensures that at least two developers know how each part of the code works.

Code release

Code releases should be a regular occurrence. Code release helps with project branding and user engagement, and ensures that the project is seen as active. It also helps project administration by ensuring new features are shared in a timely fashion, and by reducing the number of long-lived divergent branches.

The project NEPTUNE codebase contains proxyapps, and infrastructure code that interfaces them. A code release will consist of a version of this infrastructure code, plus commit hashes that fix the versions of the proxyapps.

As proxyapps might be independent projects with their own established release cycle, the following release policy applies only to the infrastructure code. It is the recommended policy for new proxyapps written under Project NEPTUNE.

Release numbering should follow (a modified) Semantic Versioning approach **semverwebsite**, summarized as

> "Given a version number MAJOR.MINOR.PATCH, increment the:
>
> 1. MAJOR version when you make incompatible API changes,
> 2. MINOR version when you add functionality in a backwards compatible manner, and
> 3. PATCH version when you make backwards compatible bug fixes."

Here it is understood that "API" refers to user-facing interfaces; APIs to functions internal to proxyapps may break backwards compatibility in MINOR releases. There is however an absolute guarantee that no backwards incompatible changes are made for end users in MINOR and PATCH releases, except those that arise from fixing a bug. That is, physics results are permitted to change in such releases if the new release's results are "correct" and the previous release's results were "wrong".

It is also understood that releases with MAJOR number 0 are considered beta releases, for which there are no guarantees of backwards compatibility.

Each release will be uploaded as a Zenodo**zenodowebsite** record with its own DOI. This gives a clear citation for the project (to be included in the project's README or CITATION.cff file), while ensuring that developers receive credit for their work, without the need for associating each release with a publication. Encouraging researchers to use release versions and to cite by version number also aids scientific reproducibility.

While some technical aspects of the release process can be automated, many of the tasks, such as curating issues and writing release notes, are inherently manual. To prevent NEPTUNE relying on a single person to make releases, the exact workflow will be codified and included in the project documentation. An example of such a workflow for the GS2 project may be found online **gs2website**.

7.3 ExCALIBUR Project NEPTUNE Charter

All members of the ExCALIBUR NEPTUNE team should be aware that to meet the challenges of the NEPTUNE project, and the ExCALIBUR overarching pillars, a distributed team of scientists, software engineers and architecture specialists from different UK institutions will be required to form a community around the NEPTUNE project (and will connect across the overarching ExCALIBUR programme). A high-level objective is to ensure that developed software is of the highest quality, implying a rigid requirement around the production of high-quality documentation and reproducible verification and validation tests for the codebase as it evolves. Since development work may transfer between institutions, it is important that common standards for documentation and testing be available and easy to deploy. The initial NEPTUNE exploratory Proxyapps may be written in a range of languages including for example Python, C++/DPC++, Object Fortran or Julia, however it is envisaged that there will be an emerging steer towards a reduced set of languages and technologies to ensure interoperability across the NEPTUNE software stack, ultimately leading to coupled simulations covering all the physics necessary to deliver an "actionable" simulation for the plasma edge. It is not yet clear for example whether SYCL, Kokkos or OpenMP 5 will offer the most performance portable and sustainable solution for NEPTUNE. The team is therefore expected to be agile and amenable to change once it is clear which are the most promising long–term solutions. For example, a selection of SYCL for the long-term framework/code(s) would force refactoring of any code that is not consistent with a NEPTUNE library and code base instantiated in DPC++, and where feasible, team members should support this process.

Source code for all development should be accessible by the entire NEPTUNE team and all tests should be repeatable by different workers without the need for re-training and/or any possible confusion as to the procedures and metrics needed to declare a test successful.

NEPTUNE will be developed as a sequence of 'core' Proxyapps (to be distinguished from other Proxyapps designed to test some novel technique). Core Proxyapps will all need a documentation and testing framework, which must be agreed between all partners for the entire project. This will require developers to work closely with UKAEA and other team members.

A commitment is also expected by all parties to help UKAEA and the Met Office (as SRO for ExCALIBUR) to publicise the project and build a fully connected community across the ExCALIBUR programme, UKRI and Academia, focused upon a team of approximately twenty UK Fusion use case experts. This will be essential for meeting the grand challenge goal of developing a state-of-the-art, Exascale targeted, UK-based plasma physics simulation capability for the tokamak edge plasma (see Science Plan **sciplan**).

All Core Proxyapps and related infrastructure/documentation across the NEPTUNE project should meet the demands of the Code Structure and Co-

ordination work package FM-WP4 in so far as the developing project standards:

- adopt a consistent choice of definitions (ontology) of objects or equivalently classes,
- adhere to clearly defined common file formats and interfaces to components for data input and output.
- provide suitably flexible data structures for common use by all developers,
- are established through good scientific software engineering best practice,
- demonstrate performance portability and exploit agreed DSL-like interfaces where possible targeting Exascale-relevant architectures,
- can be integrated into a VVUQ framework and
- are embedded within a coordination and benchmarking framework for correctness testing and performance evaluation.

In order to meet Strategic Priorities Fund terms around eligibility, and to steer the project towards a modular platform where developments across all partners can be integrated into an eventual code or platform available for open use by the European fusion community, a requirement is that all ExCALIBUR NEPTUNE Grant beneficiaries make technology / source code developed through the programme (foreground Intellectual Property) available as open source under a programme–wide permissive license (currently selected as 3-clause BSD for core/foundational infrastructure **bsd3clause**). Government Digital Service guidance (to which the project subscribes), discussing the benefits of open versus closed technology/software/data can be found in refs **os**, **os2**.

Chapter 8

Maintenance File

The Maintenance File (MF) is a developer generated file that describes the planning and status of the maintenance, migration and retirement activities.

Chapter 9

Operational Documentation

The Operational Documentation (OP) consists of the Developer Manual Section **??** and the User Manual Section **??**. It is important that the user's experience of the software feeds back into the instructions as to how to use the software, and mechanisms for achieving this end appear in Section **??**.

9.1 Documentation Generally

Documentation

Documentation refers to a particular version of the code. It should therefore be dynamic, under version control, and tightly coupled to the source code itself.

- All new code features should be documented, and this should be checked as part of the peer review process.
- Within the code, comments should use a convention, such as that accepted by DOXYGEN, to document the intent of functions, and any assumptions on their environment, input or outputs.
- Alongside the code `README` files explaining the file/directory layout typically use the Markdown format due to its simplicity, standardising on the variant defined by PANDOC as described in **y2d34**.
- The more formal documentation should be in a format which can include elements such as equations, code blocks, graphs and figures. It should also be easily convertible to other formats, and in particular online documentation. LaTeX as used to produce the current document can be easily converted to .html as explained in ref **y2d34** provided the restrictions (as to accepted packages) noted in the reference are observed.

9.2 Developer Manual

Testing

Requirements before merging changes in `git` include:

- Tests must pass. (Merge blocking can be enforced eg. on `github`.)
- The code must be in the standard style, which will be at least partly enforced as part of the automated testing.
- Documentation must be updated or added to reflect changes in the code.

Source code testing

Testing of code is essential to ensure correctness, reduce incidents of accidental breakage or regression of code features, and enable code changes to be made with confidence. These tests must be automated, and as far as possible be "unit" tests, which test isolated components of the code. A strictly Test Driven Development (TDD) approach is not always appropriate, but encouraging incremental development and testing in small pieces has several advantages in terms of the resulting code structure and maintainability:

1. It encourages the writing of code which has "clean" interfaces ie. a well defined set of inputs and outputs, with minimal side channels (eg. global state).
2. Having to test components individually discourages strong coupling between code, because then these dependency components have to be "mocked" up in testing.
3. Good code test coverage makes later maintenance, modification and refactoring of the code easier. The tests also function as a type of documentation of the intended use of the code, and also of the corner-cases which may not be obvious to a new user or developer.

The most important types of tests are for correctness. These can use standard services such as `github actions`, `Travis` etc. Performance is however a crucial property of the code, and should also be monitored.

Performance testing

It is useful to include timing information in test output, which is then contained in the test logs. This is valuable as a quick way for developers to observe the impact of changes on performance. It is however not very accurate, especially under virtual machines on shared hardware as is typical for testing services. These tests also only typically use a small number of processors (less than four),

usually without accelerator support, making them of limited use in evaluating performance of high performance code for the Exascale.

Periodic testing of code versions on a range of hardware will be needed to monitor performance, and catch performance regressions. This could be carried out by a researcher, but the possibility of automating this process and making use of services such as Amazon AWS HPC and GPU servers. Studies carried to date indicate a lack of appropriate software for ensuring performance portability and a consequent need at least to enhance existing packages.

Object Identification

Douglass **douglass** has a description of object analysis which is well-suited project NEPTUNE. His approach is to take the use cases, which for this purpose should include the proxyapps separately, and treat them carefully one after the other using the strategies indicated in Table **??**. Each proxyapp should be carefully analysed and classes produced from the list of objects before proceeding to the next.

9.3 User Manual

ICD Interface Control Document (User Manual)

With video tutorial, quick start, installation, examples. Examples use Jupyter - has Python and Julia interfaces. Binder shares a Jupyter notebook so recipient can immediately execute in a browser

Table 9.1: Key Strategies for Object Identification. After Table 5.1 from ref **douglass**, slightly amended. All the strategies except the last, are concerned with identifying the objects listed.

Strategy	Description
Nouns	Used to gain a first-cut object list, the analyst underlines each noun or noun phrase in the problem statement and evaluates it as a potential object, class, or attribute.
Causal agents	Identify the sources of actions, events, and messages; includes the coordinators of actions.
Services (passive contributors)	Identify the targets of actions, events, and messages as well as entities that passively provide services when requested.
Messages and information flow	Messages must have an object that sends them and an object that receives them as well as, possibly other objects that process the information contained in the messages. There are many ways to identify the objects within a collaboration.
Real-world items	Real-world items are entities that exist in the real world, but are not necessarily electronic devices. Examples include objects such as gases, forces, blanket modules, etc.
Physical devices	Physical devices include the sensors and actuators provided by the system as well as the electronic devices they monitor or control. The resulting objects are almost always the interfaces to the devices. Note: this is a special kind of “Identify real-world items".
Key concepts	Key concepts may be modeled as objects. Physical theories exist only conceptually, but are critical scientific objects. Frequency bins for an on-line autocorrelator may also be objects. Contrast with the “identify real-world items" strategy.
Transactions	Transactions are finite instances of interactions between objects that persist for some significant period of time. An example is queued data.
Persistent information	Information that must persist for significant periods of time may be objects or attributes. This persistence may extend beyond the power cycling of the device.
Visual elements	User-interface elements that display data are objects within the user-interface domain such as windows, buttons, scroll bars, menus, histograms, waveforms, icons, bitmaps, and fonts.
Control elements	Control elements are objects that provide the interface for the user (or some external device) to control system behavior.
Apply scenarios	Walk through scenarios using the identified objects. Missing objects will become apparent when required actions cannot be achieved with existing objects and relations.

9.4 Feedback and Communication

- Matrix chat - Slack?
- Discourse group
- Mailing list
- Suggestion box
- Weekly community meetings

Chapter 10

Reference Material

10.1 Conventions for Report Writing

The project has strict standards in respect of conventions even in reports that contain no code, ultimately to support the 'write once, use many times' concept, so that text may be cut, edited and pasted into source code, or indeed used safely to define variable names, constraints on their values and physical dimensions. Thus, since many compilers support only ASCII characters, only ASCII is allowed in reports. Similarly, very long lines (typically arising from MS Windows wordprocessing) should be split at the ends of sentences, and preferably so that maximum line length is 120 characters. A number of tools are available in `tex/importools`. Contributors may use their style of choice for both reports (and code) provided they follow systems that allow for automatic conversion to the conventions expected in the NEPTUNE repositories, and are prepared to help construct gitlab runners and github hooks to this end. However, it will be generally found easier to use LaTeX and bibtex as indicated below.

For shorter documents it is acceptable to use Markdown in the 'dialect' defined by PANDOC, see Annex A of **y3re314**. Resulting .md files frequently convert straightforwardly to LaTeX format, using the md2tex.bash script from `tex/importools`.

General textual issues

It is helpful to use LaTeX newcommand for certain keywords where they have a special format or because the terminology is not fully established. Although some do not, most people do find variations in use of spelling, punctuation, capitalisation etc. to be irritating, and so the following conventions will be enforced where possible:

- use of LaTeX newcommand for , viz. \papp and \exc instead of explicit proxyapp or mini-app and ExCALIBUR respectively
- punctuation, viz. eg.\ rather than e.g.

- hyphenation, generally to be avoided because LaTeX may use it to break lines, viz. finite element rather than finite-element (or simply FE), open-source rather than open source, major exception is abbreviation of dimensions (below).

- capitalisation, eg. Exascale rather than exascale

- abbreviation of dimensions, 2-D and 3-D preferred to 2D and 3D or 2d and 3d

- spelling, UK English preferred, and '-ise' etc. preferred to '-ize'

- no spaces between authors' initials in bibtex files (see also Section **??**)

- consistent usage of acronyms, employing the table of Section **??**

- consistent usage of mathematical symbols, employing the table of Section **??**

Citations

Regarding citations, those for unpublished reports MUST include a link to a website, such as arXiv, and it would be helpful if links to other open-access material were also given. Use of the following conventions for constructing the keys of citations should help avoid clashes:

1. For published papers, use where possible an 8-character alphanumeric for published papers, consisting of the first two letters of the first author's name, the last two digits of the year in the Gregorian calendar, and the first 4 letters of the first *significant* word (ie. not 'The' or 'On') of the title, preserving capitalisations. Thus the paper **Ba13What** by Bangerth and Heister, published in 2013 and entitled 'What makes computational open source software libraries successful?', has key 'Ba13What'.

2. Books and theses should be keyed with the full second-name(s) of the author(s) strung together without capitalisation, up to a limit of approx. 30 characters, using 'etal' to indicate any omitted author-names. As an example, the book by Rouson, Xia and Xu **rousonxiaxu** has key 'rousonxiaxu'.

3. If there are duplicates in different files, then preface each key with the name of the .bib file it is in, for other duplicates within a file, add '2', '3' etc. to the end of the key.

Software Compatibility

When discussing software in the text, the following conventions in LaTeX are proposed:

- SMALL CAPITALS denote a package name, use {\textsc or abbreviation \F{
- *Italics* denote a program name, use {\textit or abbreviation \I{
- `Fixed width font` denotes any code name or fragment which is not otherwise obviously source code, use {\texttt or abbreviation \T{

There is no need for special fonts if the object is identified by a suitable suffix, thus "_m" for a module containing executable code, "_h" or ".h" for an object description or namespace code, ".cpp" for name of file containing C++ source, ".exe" for an executable, etc. Similarly file suffices that imply a particular format or software for their interpretation, may simply be written with a leading stop, eg. ".html" and ".exe".

In order to ensure smooth transliteration from mathematical symbols to the names of the software variables, Table **??** lists the recommended two-character equivalents for LaTeX symbols used in the definition of mathematical symbols.

Table 10.1: **TWO CHARACTER EQUIVALENTS** of LaTeX symbols and commands.

aa	A	al	α	ar	$\rightarrow$
as	$*$	bb	B	be	β
bl	$[$	br	$]$	cc	C
ch	χ	ci	$\hat{}$	dd	D
de	δ	dl	Δ	dq	"
ds	\ddot	dv	$/$	ee	E
el	ℓ				
ep	ϵ	et	η	ff	F
ga	γ	gg	G	gm	Γ
gt	$>$	hh	H	ii	I
in	∞				
it	ι	jj	J	ka	κ
kk	K	la	λ	ll	L
lm	Λ	lt	$<$	me	ω
mg	Ω	ml	$\times$	mm	M
mn	$-$	mu	μ	n1	n_1 etc.
nn	N	nu	ν	o2	\boldmath
o3	\mathcal	o4	\mathsf	o5	\mathtt
o6	\mathbb	o7	\mathfrak		
oe	suffix	ol	preceding superfix	on	above
or	superfix	os	underneath	ow	prefix
pa	$\|\|$	pe	$\perp$	pf	Φ
ph	ϕ	pi	π	pj	Ψ
pl	$\{$	pp	P	pr	$\}$
ps	ψ	pt	∂	pu	$+$
py	Π	qq	Q	rh	ρ
rr	R	sg	Σ	si	σ
sq	$'$	ss	S	st	.
ta	τ	te	Θ	th	θ
ti		tt	T	un	_
up	υ				
us	Υ	uu	U	vb	$\|$
ve	ε	vf	φ	vp	ϖ
vr	ϱ	vs	ς	vt	ϑ
vv	V	ww	W	xx	X
yy	Y	ze	ζ	zz	Z

See Section **??** for a guide explaining the reasons for the above choices.

10.2 Acronyms

Table 10.2: **TABLE OF ACRONYMS** Nearly all the acronyms refer to technical terms. A debt is acknowledged to the book by Brunton and Kutz **bruntonkutz**.

ACM	Association for Computing Machinery
ADC	Analogue to digital converter
ADM	Alternating directions method
AIC	Akaike information criterion
ALM	Augmented Lagrange multiplier
AMR	Adaptive mesh refinement
AMReX	Software framework for block-structured AMR
ANL	Argonne National Laboratory
ANN	Artificial Neural Network
ANOVA	Analysis of Variance
API	Application Programming Interface
ARMA	Autoregressive moving average
ARMAX	Autoregressive moving average with exogenous input
ASQ	Adaptive sparse quadrature
ATS	Advanced Terrestrial Simulator, previously Arctic Terrestrial Simulator
BC	Boundary Condition
BEIS	(UK government) Department for Business, Energy and Industrial Strategy
BG/L	IBM Blue Gene / L supercomputer platform
BIC	Bayesian information criterion
BOUT++	Tokamak edge plasma modelling framework `https://boutproject.github.io/`
BPOD	Balanced proper orthogonal decomposition
BSD	Opensource software licence
CAD	Computer-Aided Design, geometry including NURBS, usually "CAD database" implied
CCA	Canonical correlation analysis
CCFE	Culham Centre for Fusion Energy
CEA	The French Alternative Energies and Atomic Energy Commission
CESM	Community Earth System Model
CFD	Computational fluid dynamics
CI	Continuous integration
CLI	Command Line Interface
CNN	Convolutional neural network
COGENT	LLNL continuum plasma simulation code
COMPAT	Computing patterns for multiscale HPC (project)
CoSaMP	Compressive sampling matching pursuit
COSMO	Framework for regional weather prediction in Europe

COSSAN	UQ and risk analysis package (Uni. Liverpool)
CPP	C plus plus programming language
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
CS	Compressed sensing
CSE	Computational science and engineering
CSG	Constructive Solid Geometry
CSMP	Computer science, mathematics, and physics
CTO	Chief Technology Officer
CUDA	Compute Unified Device Architecture
CWIPI	Coupling with interpolation parallel interface (coupling library)
CWT	Continuous wavelet transform
DA	Data Assimilation
DAG	Direct Acyclic Graph
DAKOTA	UQ and optimization package (Sandia)
DCT	Discrete cosine transform
DDA	Digital Differential Analyser
DDD	Document-Driven Design
DE	Differential equation
DEIM	Discrete Empirical Interpolation Method
DFT	Discrete Fourier Transform
DiMDc	Dynamic mode decomposition with control
DL	Deep learning
DMD	Dynamic mode decomposition
DMDc	Dynamic mode decomposition with control
DNS	Direct numerical simulation
DOE	Department of Energy
DOI	Digital Object Identifier
DPC++	Data Parallel C++, Intel compiler for C++ with SYCL extension
DRAM	Delayed Rejection Adaptive Metropolis
DSL	Domain-Specific Language
DWT	Discrete wavelet transform
ECOG	Electrocorticography
ECP	Exascale Computing Project
ECP-copa	Co-design centre for particle applications (part of ECP)
eDMD	Extended DMD
EIM	Empirical interpolation method
EIRENE	name of neutral package
EM	Expectation maximization
EOF	Empirical orthogonal functions
ERA	Eigensystem realization algorithm
ESC	Extremum-seeking control
ESI	name of software company `https://www.esi-group.com/`
ESMF	Earth System Modeling Framework

E-TASC	EUROfusion Theory and Advanced Simulation Coordination
ETS	European Transport Simulator
EU	European Union
FCI	Flux-Coordinate Independent (method)
FELTOR	name of edge code
FEM	Finite Element Method
FEniCS	name of PDE software project `https://fenicsproject.org`
FFT	Fast Fourier Transform
FFTW	Fastest Fourier Transform in the West (library)
FLASH	name of Multiscale physics code
GA	General Atomics
GBS	Global Braginskii Solver (software)
GCR	Generalied Collisional Radiative (framework)
GDB	Global Drift-Ballooning
GDB	GNU debugger
GDPR	General Data Protection Regulation
GENE	name of gyrokinetic code
GMM	Gaussian mixture model
GMRES	Generalized Minimal Residual method
GNU	GNU's Not Unix!
GP	Gaussian Process
gPC	Generalised polynomial chaos (Xiu and Karniadakis `https://doi.org/10.1016/S0021-9991(03)00092-5`
GPU	Graphics Processing Unit
GRILLIX	name of 3D turbulence code based on the flux-coordinate independent approach
GSA	Global sensitivity analysis
GUI	Graphical User Interface
HAGIS	HAmiltonian Gulding centre System
HAVOK	Hankel alternative view of Koopman
HDF5	Hierarchical Data Format (version 5)
HDS	Hierarchical Data Structure
HLA	High Level Architecture
HPC	High Performance Computing
HTC	High Throughput Computing
IBM	International Business Machines Corp., but really known as IBM
IC	Initial Condition
ICA	Independent component analysis
ICON	ICOsahedral Nonhydrostatic, the global numerical weather prediction model of the German weather service
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Taskforce
IMAS	Integrated Modelling & Analysis Suite, promoted by ITER
IMEX	Implicit-Explicit Methods

IO	Input/Output
ITER	name of International Thermonuclear Experimental Reactor
ITG	Ion Temperature Gradient
ITM	Ion Tearing Mode
ITPA	International Tokamak Physics Activity (ITER research programme)
JET	Joint European Torus
JIT	Just In Time
JL	JohnsonLindensfrauss
JOREK	name of nonlinear MHD code
JSON	JavaScript Object Notation
KL	Kullback Leibler
KLT	Karhunen-Loeve transform
LAD	Least absolute deviations
LAMMPS	Large-scale Atomic/Molecular Massively Parallel Simulator
LANL	Los Alamos National Laboratory
LASSO	Least Absolute Shrinkage and Selection Operator
LCFS	Last Closed Flux Surface
LDA	Linear discriminant analysis
LGPL	GNU Lesser General Public License
LHSamp	Latin Hypercube Sampling
LLNL	Lawrence Livermore National Laboratory
LOO	Leave One Out
LQE	Linear quadratic estimator
LQG	Linear quadratic Gaussian controller
LQR	Linear quadratic regulator
LTI	Linear time invariant system
MAP	Maximium A Posteriori
MBSE	Model-based systems engineering
MC	Monte-Carlo (methods)
MCMC	Markov chain Monte-Carlo
MCT	Model Coupling Toolkit
MD	Molecular Dynamics
MECE	Mutually exclusive and collectively exhaustive
MF	Multi-fidelity, Matrix-free
MFMC	Multi-fidelity Monte-Carlo
MHD	Magnetohydrodynamics
MIMC	Multi-Index Monte-Carlo
MIMO	Multiple input, multiple output
MIS	Module Interface Specification
MIT	Massachusetts Institute of Technology
MIT licence	Opensource software licence **MITlicense**
ML	Machine Learning
MLC	Machine learning control
MLMC	Multi-Level Monte-Carlo

MLMF	Multi-Level Multi-Fidelity
MMF	Multiscale Modeling Framework
MMS	Method of Manufactured Solutions
MOOSE	Multiphysics Object Oriented Simulation Environment
MOR	Model Order Reduction
MPE	Missing point estimation
MPI	Message Passing Interface
mrDMD	Multi-resolution dynamic mode decomposition
MSSC	Materials Science and Scientific Computing
MUMPS	MUltifrontal Massively Parallel Sparse direct Solver
MUSCLE 3	Multiscale Coupling Library and Environment version 3
NAG	Numerical Algorithms Group
NARMAX	Nonlinear autoregressive model with exogenous inputs
NEMO	Nucleus for European Modelling of the Ocean
NEPTUNE	Neutrals and Plasma Turbulence Numerics for the Exascale
NetCDF	Network Common Data Form
NLS	Nonlinear Schroedinger equation
NROY	Not ruled out yet
NUCODE	Software: SMARDDA/NUCODE for Neutral Beam Duct Calculations
NURBS	NonUniform Rational B-Spline
OASIS	Ocean Atmosphere Sea Ice Soil
OASIS4	Ocean Atmosphere Sea Ice Soil version 4
ODE	Ordinary Differential Equation
OKID	Observer Kalman filter identification
OLYMPUS	OLYMPUS Programming System
OMFIT	One Modeling Framework for Integrated Tasks
OneAPI	A Unified, Standards-Based Programming Model, `https://software.intel.com/en-us/oneapi`
OP2	API with associated libraries and preprocessors for performance-portable parallel computations on unstructured meshes `https://github.com/OP-DSL/OP2-Common`
OpenMP	Open Multi-Processing
OU	Oxford University
OUU	Optimisation under uncertainty
PASTIX	Parallel Sparse matriX package
PBH	PopovBelevitchHautus test
PC	Polynomial chaos
PCA	Principal components analysis
PCE	Polynomial chaos expansion
PCP	Principal component pursuit
PDE	Partial Differential Equation
PDE-FIND	Partial differential equation functional identification of nonlinear dynamics
PDF	Probability distribution function

PETSc	Portable Extensible Toolkit for Scientific Computation `https://www.mcs.anl.gov/petsc/`
PFC	Plasma Facing Component
PGD	Proper Generalised Decomposition
PIC	Particle-In-Cell
PICPIF	Particle-In-Cell-Particle-In-Fourier
PID	Proportional-integral-derivative control
PIV	Particle image velocimetry
POD	Proper Orthogonal Decomposition
POOMA	Parallel Object-Oriented Methods and Applications
PP20	SIAM Conference on Parallel Processing for Scientific Computing 2020
PPMD	Performance-Portable Framework For Atomistic Simulations
PR	git Pull Request
PSyclone	PSyclone is a code generation system that generates appropriate code for the PSyKAl code structure developed in the GungHo project. `https://github.com/stfc/PSyclone`
PyOP2	Framework for performance-portable parallel computations on unstructured meshes `http://op2.github.com/PyOP2`
QA	Quality Assurance
QCG	Quality in Cloud and Grid, see QCG Pilot Job
QMC	Quasi-Monte-Carlo
QoI	Quantity of interest
QoS	Quality of Service
RAID	Risks, Assumptions, Issues, Dependencies
RAJA	RAJA Performance Portability Layer (C++) `https://github.com/LLNL/RAJA`
REST	Representational State Transfer (Resources as simple CRUD objects)
RIP	Restricted isometry property
RKF23	Runge-Kutta-Fehlberg (*aka* Embedded Runge-Kutta), 23 denotes orders of scheme
RKHS	Reproducing kernel Hilbert space
RMS	Root-mean-square
RNG	Random Number Generator
RNN	Recurrent neural network
RO	Responsible Officer
ROM	Reduced-Order Model
RPCA	Robust principal components analysis
rSVD	Randomized SVD
SAMRAI	Structured Adaptive Mesh Refinement Application Infrastructure
SD1D	name of 1-D edge code
SDLC	Software Development Life Cycle
SGD	Stochastic gradient descent
SIAM	Society for Industrial and Applied Mathematics
SINDy	Sparse identification of nonlinear dynamics

SISO	Single input, single output
SLA	Service-level Agreement
SLE	Software Language Extensions
SLE	System Level Engineering
SLEPc	name of Scalable Library for Eigenvalue Problem Computations
SLSQT	Sequential Least-Squares' Thresholding
SMARDDA	name of Ray-tracing algorithm, hybrid of SMART and DDA
SMART	name of Ray-tracing algorithm based on use of octree
SMITER	SMARDDA modules with ITER interface
SNOWPAC	Stochastic Nonlinear Optimisation with Path-Augmented Constraints (software package)
SOL	Scrape-Off Layer
SOLEDGE	name of edge modelling code
SOLPS	name of edge modelling code combines B2 and EIRENE
SPH	Smoothed Particle Hydrodynamics
SRC	Sparse representation for classification
SRO	Senior Responsible Owner role in UK government project delivery
SRS	Software Requirements Specification
SSA	Singular spectrum analysis
SSD	Scientific Software Development
StarPU	Runtime system supporting heterogeneous multicore architectures `http://starpu.gforge.inria.fr/doc/html/`
STARWALL	name of vacuum field code
STFT	Short time Fourier transform
STIX	Scientific And Technical Information eXchange
STLS	Sequential thresholded least-squares
STORM	Scrape-off layer Transport ORiented Module
STRUMPACK	STRUctured Matrix PACKage
SUNDIALS	name of ODE package
SVD	Singular Value Decomposition
SVM	Support Vector Machine
SYCL	C++-single-source heterogeneous programming for acceleration offload, `https://www.khronos.org/sycl/`
SysML	Systems Modeling Language
TAE	Toroidal Alfven Eigenmode
TDD	Test Driven Development
TICA	Time-lagged independent component analysis
TM	TradeMark
TOKAM	name of set of edge modelling codes
TOKAM3X	name of Edge modelling software
TOMS	Transactions on Mathematical Software
TORPEX	TORoidal Plasma Experiment

Trilinos	Object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems `https://trilinos.github.io/`
TRIMEG	TRIangular MEsh based Gyrokinetic code
TSVV	Theory, Simulation, Validation and Verification, tasks of the E-TASC programme of Eurofusion
TUM	Technical University Munich
UK	United Kingdom
UKAEA	United Kingdom Atomic Energy Authority
UKRI	United Kingdom Research and Innovation, a non-departmental public body encompassing the research councils and Innovate UK
UML	Unified Modelling Language
UQ	Uncertainty quantification
US	United States
USA	United States of America
UTF-8	Unicode Transformation Format (Unicode denotes Universal Coded Character Set)
UUID	Universally Unique IDentifier is a 128-bit label used for information in computer systems
VAC	Variational approach of conformation dynamics
VDE	Vertical Dispacement Event
VECMA	Verified Exascale Computing for Multiscale Applications
VECMAtk	VECMA toolkit
VORPAL	name of Electromagnetic Particle-in-Cell code
VSVO	variable stepsize, variable order solver of differential equation
VVUQ	Verification, Validation and Uncertainty Quantification
XGC1	name of Particle-based gyrokinetic code
XML	eXtensible Markup Language
XMSF	eXtensible Modeling and Simulation Framework
XPN	ExCALIBUR Project NEPTUNE

10.3 Symbols

Table 10.3: **TABLE OF MATHEMATICAL SYMBOLS** If no units are given, then quantity is dimensionless, or if the units are given as $?$, then the dimensions depend on context. Generally, the usage of symbols tries to follow that from the Plasma Formulary **NRLpf07**, in SI units, with temperatures specified as kT which returns J. The Formulary also give the fundamental dimensions of the SI units, which should enable checking of dimensional consistency of equations, eg. magnetic field induction is in Tesla (T) whence the fundamental dimension expression gives $T = kgs^{-1}C^{-1}$. Note that the symbols are sorted by font as well as alphabet, so that boldface symbols appear immediately after 'b' (backslashes ignored). The main source for the symbols is the Equations document **pappeqs**, also included are those listed as used in the text by Karniadakis and Sherwin **karniadakissherwin**, prefaced by (K+S), plus symbols used in the report **y2re313**.

Symbol	Description	Units
a	minor radius of the torus (horizontal)	m
A	atomic mass of the ion	
$[a, b]$	arbitrary finite interval	
α	as suffix is species label or index	
α_n	perturbation amplitude	
$\alpha^{Z_p \to Z}$	partial dielectronic recombination rate coefficient	$m^3 s^{-1}$
$\alpha^{Z \to Z_m}$	partial dielectronic recombination rate coefficient	$m^3 s^{-1}$
b	minor radius of the torus (vertical)	m
B_0	used to make B dimensionless	T
$\bar{N}_Z$	average number of charge states	
$B = \|\mathbf{B}\|$	amplitude of the imposed magnetic field	T
β	as suffix is species label	
β	(Glossary) Ratio of plasma pressure to pressure in magnetic field	
$\mathbf{a} = d^2\mathbf{x}/dt^2$	acceleration experienced by a particle	$m^2 s^{-1}$
$\mathbf{A}(\mathbf{x}, t)$	magnetic vector potential	Tm
$\mathbf{B}(\mathbf{x}, t)$	magnetic field	T
$\mathbf{b}$	unit vector giving the direction of the magnetic field	
$\mathbf{E}(\mathbf{x}, t)$	electric field	Vm^{-1}
$\mathbf{E}^+$	modified electric field	m^{-2}
$\mathbf{F}$	force vector	N
$\mathbf{u}_0$	initial fluid velocity	ms^{-1}
$\mathbf{u}_{cx}$	'charge exchange' perpendicular fluid velocity component	ms^{-1}

$\mathbf{u}_{E\times B}$		'E cross B' perpendicular fluid velocity component	ms^{-1}
$\mathbf{u}_e$		velocity of the electrons	ms^{-1}
$\mathbf{u}_i$		velocity of the ion species	ms^{-1}
$\mathbf{u}_{\nabla Be}$		'grad B' perpendicular fluid velocity component for electrons	ms^{-1}
$\mathbf{u}_{\nabla Bi}$		'grad B' perpendicular fluid velocity component for ions	ms^{-1}
$\mathbf{u}_{\text{diff}}$		'diffusive' perpendicular fluid velocity component	ms^{-1}
$\mathbf{v}_\alpha$		velocity of species α	ms^{-1}
$\mathbf{v}_e$		velocity of the electrons	ms^{-1}
$\mathbf{v}_\parallel$		fluid velocity component along fieldline	ms^{-1}
$\mathbf{v}$		generic velocity	ms^{-1}
$\mathbf{v}_i$		velocity of the ion species	ms^{-1}
$\mathbf{v}_\perp$		fluid velocity component normal to flux surface	ms^{-1}
$\mathbf{v}_\wedge$		fluid velocity component in flux surface normal to field direction	ms^{-1}
$\mathbf{x}$	$=$	is a d-dimensional vector	
$(x_1, x_2, \ldots, x_d)$			
$\mathbf{x}$		position	m
b_n		'b-factors' ref **omullane**	
$\xi(\theta)$		multi-dimensional random variable with a specific probability distribution as a function of the random parameter $0 \leq \theta \leq 1$	
$\boldsymbol{B}$		(K+S) Basis matrix	
$\boldsymbol{D}_\xi$		(K+S) Elemental derivative matrix with respect to ξ	
$\boldsymbol{f}^e$		(K+S) Force vector of the eth element	
$\boldsymbol{H}$		(K+S) Helmholtz matrix ($= \mathcal{A}^T \underline{\boldsymbol{H}^e} \mathcal{A}$))	
$\boldsymbol{H}^e$		(K+S) Elemental Helmholtz matrix	
$\boldsymbol{L}$		(K+S) Laplacian matrix ($= \mathcal{A}^T \underline{\boldsymbol{L}^e} \mathcal{A}$))	
$\boldsymbol{\Lambda}(u)$		(K+S) Diagonal matrix of $u(\xi_i, \xi_2)$ evaluated at quadrature points	
$\boldsymbol{L}^e$		(K+S) Elemental Laplacian matrix	
$\boldsymbol{M}$		(K+S) Mass matrix ($= \mathcal{A}^T \underline{\boldsymbol{M}^e} \mathcal{A}$)	
$\boldsymbol{\mathcal{A}}^T$		(K+S) Matrix global assembly	
$\boldsymbol{M}^e$		(K+S) Elemental mass matrix	
$\boldsymbol{n}$		(K+S) Unit outward normal	
$\boldsymbol{\omega}$		(K+S) Vorticity	
$\boldsymbol{u}^e$		(K+S) Vector containing function evaluated at quadrature points	
$\boldsymbol{W}$		(K+S) Diagonal weight / Jacobian matrix	
B_p		amplitude of the poloidal magnetic field	T
B_T		amplitude of the imposed toroidal magnetic field	T
$C_0 = \sqrt{kT_0}$		used to make velocities dimensionless	ms^{-1}
$\cap$		(Sets) Set intersection	
χ		(K+S) Space of trial solutions	
χ^δ		(K+S) Finite-dimensional space of trial solutions	
$\chi_i(\xi)$		(FE Basis) Local Cartesian to global coordinate mapping	

c_p	specific heat at constant pressure	$Jkg^{-1}K^{-1}$
$c_s = \sqrt{\frac{kT_i + Z_i kT_e}{m_i}}$	acoustic speed	ms^{-1}
$c_{se} = \sqrt{\frac{kT_e}{m_e}}$	acoustic speed of electrons	ms^{-1}
$c_{si} = \sqrt{\frac{kT_i}{m_i}}$	acoustic speed of ions	ms^{-1}
C_S	sound speed coefficient in radiation equation	ms^{-1}
$\cup$	(Sets) Set union	
$C(x_i, x_j)$	covariance of random variables x_i, x_j	
d	number of dimensions over which the integral is performed	
δp_i	stress tensor	Nm^{-2}
δ	Kronecker delta function	
δ_e	energy flux factor at boundary of the electrons	
$\delta = \frac{1}{2}(\delta_e + \delta_i)$	energy flux factor at boundary of 'mean' species	
δ_i	energy flux factor at boundary of the ion species	
δ_s	(Glossary) Magnetisation parameter, species s gyroradius normalised to L	
$\delta(x)$	Dirac delta function of continuous real variable x	
D_A	diffusion coefficient for plasma charges in a background of neutrals	m^2s^{-1}
D_e	diffusion coefficient for electrons in a background of neutrals	m^2s^{-1}
D_n	neutral diffusion coefficient	m^2s^{-1}
D_i	diffusion coefficient for ions in a background of neutrals	m^2s^{-1}
$\|e\|$	absolute value of the charge on the electron	C
e	(K+S) Finite element number $1 \leq e \leq N_{el}$	
e_{ijk}	weighted integral of triple products of Ψ_i of the ion species	
$\emptyset$	(Sets) Empty set	
ϵ_0	permittivity of free space	Fm^{-1}
$\epsilon_r = L_s/(t_0 C_0)$	scale factor for transient term	
η_1, η_2, η_3	(FE Basis) Local collapsed Cartesian coordinates	
η_B	plasma resistivity after Braginskii	Ωm
$\eta_d = \eta_B/\mu_0$	plasma resistivity, as diffusivity	m^2s^{-1}
$\eta_{e\mathsf{n}}$	contribution to plasma resistivity, as diffusivity, from electron-neutral interactions	m^2s^{-1}
$\eta_{e\mathsf{n}\parallel}$	contribution to plasma parallel resistivity, as diffusivity, from electron-neutral interactions	m^2s^{-1}
$\eta_{i\mathsf{n}}$	contribution to plasma resistivity, as diffusivity, from ion-neutral interactions	m^2s^{-1}
$\eta_{i\mathsf{n}\parallel}$	contribution to plasma parallel resistivity, as diffusivity, from ion-neutral interactions	m^2s^{-1}
f_0	constant in the expansion of $f(x_1, \ldots, x_d)$	
f_0	initial distribution function of the electrons	$m^{-6}s^3$
f_α	distribution function of species α	$m^{-6}s^3$
f_e	distribution function of the electrons	$m^{-6}s^3$

f_i	distribution function of the ion species	$m^{-6}s^3$
$f_{ij}(x_i, x_j)$	coefficient in the expansion of $f(x_1, \ldots, x_d)$	
$f_{ce} = \frac{\omega_{ce}}{2\pi}$	electron cyclotron frequency	s^{-1}
$f_{ci} = \frac{\omega_{ci}}{2\pi}$	ion cyclotron frequency	s^{-1}
$f_{pe} = \frac{\omega_{pe}}{2\pi}$	electron plasma frequency	s^{-1}
$f_{pi} = \frac{\omega_{pi}}{2\pi}$	ion plasma frequency	s^{-1}
$f_i(x_i)$	coefficient in the expansion of $f(x_1, \ldots, x_d)$	
$f(x_1, \ldots, x_d)$	joint probability distribution	
$f^{\mathcal{E}}$	flux term (fieldline integrated source) for plasma energy	
$F^{\mathcal{E}}$	flux term (fieldline integrated source divided by field) for plasma energy	$m^{-1}s^{-2}C$
f^n	flux term (fieldline integrated source) for plasma number density	
F^n	flux term (fieldline integrated source divided by field) for plasma number density	$m^{-3}C$
f^u	flux term (fieldline integrated source) for plasma momentum	
F^u	flux term (fieldline integrated source divided by field) for plasma momentum	$m^{-2}s^{-1}C$
$f(x, \mathbf{v}, t)$	generic distribution function	$m^{-6}s^3$
$\Gamma(x)$	gamma function of continuous variable x	
$g(h_j)$	activation function (of input h_j) of a neuron in a neural network	
H_α	Hamiltonian for species α	
$\hat{\boldsymbol{u}}^e$	(K+S) Vector of expansion coefficients	
$\hat{v}_g$	(K+S) Global list of coefficients	
$\hat{v}_g$	(K+S) List of all elemental coefficients ($= \underline{v^e}$)	
h_j	real-number input to a neuron in a neural network	
$h_p(\xi)$	(FE Basis) One-dimensional Lagrange polynomial of order p	
i	as suffix denotes ions	
i	as suffix denotes regular excited state	
i	as suffix generic label	
I_ϕ	$\phi-$ or toroidal component of plasma current	A
I_H	Hydrogen reionisation potential as defined in ref **Ha13Benc**	eV
i, j, k	(K+S) General summation indices	
${}^I\mathcal{F}_{i\sigma}$	coefficient of ionisation for the transition from metastable state σ to regular excited state i	
$\in$	(Sets) Is a member of; belongs to	
$I(\psi) = B_T/R$	function giving the toroidal field as a function of ψ	Tm^{-1}
I^Z	power per atom released in dielectronic recombination	W
j	as suffix is generic label	
$j_{ext}(R, Z)$	electric current density induced in plasma by external coils	Am^{-2}
j_ϕ	$\phi-$ or toroidal component of plasma current density	Am^{-2}
k	as suffix is generic label	
k	chosen to scale so that kT_0, kT_d is an energy	?
κ_α	thermal diffusivity of species α	m^2s^{-1}

$\kappa_{e\|}$	parallel thermal diffusivity of electrons	m^2s^{-1}
$\kappa_{e\perp}$	perpendicular thermal diffusivity of electrons	m^2s^{-1}
$\kappa_{i\|}$	parallel thermal diffusivity of ions	m^2s^{-1}
$\kappa_{i\perp}$	perpendicular thermal diffusivity of ions	m^2s^{-1}
$\kappa = k_c/\rho_m c_p$	thermal diffusivity tensor of solid	m^2s^{-1}
k_B	Boltzmann's constant	JK^{-1}
k_c	thermal conductivity tensor	$Jm^{-1}s^{-1}K^{-1}$
$K_{cx}\left(n_i, T_i\right)$	reaction rate of charge exchange reactions	m^3s^{-1}
K_i	ionization reaction rate	m^3s^{-1}
K_M	chosen as k_B/m_i or $\|e\|/m_i$ so that $\sqrt{K_M T_0}$, $\sqrt{K_M T_d}$ is an ion speed	?
K_r	recombination reaction rate	m^3s^{-1}
kT_0	T_0 in energy units	J
kT_d	T_d in energy units	J
$K_v(x)$	modified Bessel function of the second kind, order v	
k_w	wavenumber vector	m^{-1}
λ	arbitrary quantity	?
λ	Coulomb logarithm	
λ	(K+S) Helmholtz equation constant	
Λ	Coulomb logarithm	
λ_q	e-folding length of midplane profile of power loss when an exponential is fitted	m
Λ_b	sheath potential drop normalized to T_e	eV
λ_D	(Glossary) Debye lengthscale above which local electrostatic fluctuations due to presence of discrete charged particles are negligible	m
λ_{smfp}	(Glossary) Mean free path of particle species s	m
$\langle\sigma v\rangle_{CX}$	reaction rate for charge exchange	m^3s^{-1}
$\langle\sigma v\rangle_{ION}$	reaction rate for ionisation	m^3s^{-1}
$\langle\sigma v\rangle_{REC}$	reaction rate for recombination	m^3s^{-1}
$\langle\sigma v\rangle$	generic reaction rate	m^3s^{-1}
L_0	Typical lengthscale	m
$L_i^{N_m}(\boldsymbol{\xi})$	(FE Basis) Two-dimensional Lagrange polynomial through N_m nodes ξ_i	
L_s	length of fieldline	m
m	species particle mass	kg
M_0	Mach number at $s = 0$ boundary	
M_1	Mach number at $s = 1$ boundary	
m_α	mass of species α	kg
$\mathbb{E}$	expectation	
$\mathbb{E}_{k\neq i, l\neq j}$	expectation computed by integrating over all the x_k except for x_i and x_j	
$\mathbb{E}_{x_{k\neq i}}$	expectation computed by integrating over all the x_k except for x_i	

$\mathbb{L}(u)$	(K+S) Linear operator in u	
$\mathbb{P}$	(K+S) Projection operator	
$\mathbb{P}^\delta$	(K+S) Discrete projection operator	
$\mathbf{v}$	(K+S) Velocity $[u, v, w]^T$	
$\mathcal{E}_\alpha$	energy of species α	Jm^{-3}
$\mathcal{E}_e$	energy of the electrons	Jm^{-3}
$\mathcal{E}_i$	energy of the ion species	Jm^{-3}
$\mathcal{E}_R$	total plasma radiation	Wm^{-3}
$\mathcal{F}$	generic coefficient of excitation, ionisation or recombination	m^3s^{-1}
$\mathcal{F}_\alpha$	functional of moments of species α	$m^{-6}s^3$
$\mathcal{I}$	(K+S) Interpolation operator	
$\mathcal{I}^\delta$	(K+S) Discrete interpolation operator	
$\mathcal{K}_\parallel$	parallel thermal conductivity of plasma	$m^{-1}s^{-1}$
$\mathcal{K}$	thermal conductivity of plasma	$m^{-1}s^{-1}$
$\mathcal{K}_\perp$	thermal conductivity of plasma perpendicular to field and flux surface	$m^{-1}s^{-1}$
$\mathcal{K}_\wedge$	thermal conductivity of plasma perpendicular to field in flux surface	$m^{-1}s^{-1}$
$\mathcal{L}_7$	7-D Lie derivative (space, velocity-space and time make up the $3+3+1=7$ dimensions)	s^{-1}
$\mathcal{P}_P(\Omega)$	(K+S) Polynomial space of order P over Ω	
$\mathcal{Q}$	coefficient in radiation equation	m^3s^{-1}
$\mathcal{Q}^{Z\to Z}_{\sigma\to\rho}$	parent-metastable cross-coupling coefficient	m^3s^{-1}
$\mathcal{S}$	coefficient in radiation equation	m^3s^{-1}
$\mathcal{S}^{Z_m\to Z}$	ionisation coefficient	m^3s^{-1}
$\mathcal{S}^{Z\to Z_p}$	ionisation coefficient	m^3s^{-1}
$\mathcal{T}$	generic tensor	?
$\mathcal{V}$	(K+S) Space of test functions	
$\mathcal{V}^\delta$	(K+S) Finite-dimensional space of test functions	
$\mathcal{X}$	coefficient in radiation equation	m^3s^{-1}
$\mathcal{X}^{Z\to Z}_{\sigma\to\rho}$	generalised collisional-radiative (GCR) excitation coefficient	m^3s^{-1}
$\mathfrak{R}$	Real numbers	
$\mathrm{Var}(f)$	variance of the distribution of f computed by integrating over all variables x_i	
$\mathrm{Var}[Q]$	variance in random variable Q	
m_e	mass of electron	kg
m_i	mass of ion species particle	kg
m_n	neutral species particle mass	kg
m_p	mass of proton	kg
M_s	Mach number, allowed to take either sign	
M_S	number of energy states of an atom	
μ, ν	(K+S) Dynamic, kinematic viscosities	
$\mu_{cx} = \omega_c/\nu_{cx}$	measures strength of magnetization with respect to charge exchange reaction	

M_Z	number of metastable states for species s (which includes the ground state)	
n	number density	m^{-3}
N	number density, may be in units of $10^{18}\,\mathrm{m}^{-3}$	m^{-3}
n_0	initial number density	m^{-3}
$\nabla\cdot$	(K+S) Divergence	
$\nabla\times$	(K+S) Curl	
∇^2	(K+S) Laplacian	
N_b	(K+S) Number of global boundary degrees of freedom	
$n_B = N/B$	number density divided by field strength	$m^{-3}T^{-1}$
N_{dof}	(K+S) Number of global degrees of freedom	
n_e	number density of the electrons	m^{-3}
N_{el}	(K+S) Number of finite elements	
N_{eof}	(K+S) Total number of elemental degrees of freedom $N_{eof} \simeq N_{el}N_m$	
n_i	number density of the plasma ions	m^{-3}
$n_j(\mathbf{x}, t)$	member of the set of deterministic coefficients of the "random trial basis"	
N_m	(K+S) Number of elemental degrees of freedom	
n_n	neutral density	m^{-3}
$\notin$	(Sets) Is not a member of; does not belong to	
$\not\subset$	(Sets) Is not a subset of	
n_p	number density of the plasma ions	m^{-3}
N_Q	(K+S) Total number of quadrature points $N_Q = Q_1Q_2Q_3$	
n_s	number density of isotope s	m^{-3}
N_s	number density of isotope s	m^{-3}
N_T	number of samples in temperature used to define typically a crossection in the ADAS database **adaswebsite**, **openadaswebsite**	
ν	plasma kinematic viscosity	m^2s^{-1}
ν_α	kinematic viscosity of species α	m^2s^{-1}
$\nu_{cx} = K_{cx}n_n$	charge exchange 'frequency'	s^{-1}
ν_{e0}	electron kinematic viscosity caused by neutrals	m^2s^{-1}
$\nu_{e\|\|}$	parallel kinematic viscosity of electrons	m^2s^{-1}
$\nu_{e\perp}$	perpendicular kinematic viscosity of electrons	m^2s^{-1}
$\nu_{i\|\|}$	parallel kinematic viscosity of ions	m^2s^{-1}
ν_i	ion kinematic viscosity	m^2s^{-1}
ν_{i0}	ion kinematic viscosity caused by neutrals	m^2s^{-1}
$\nu_{i\perp}$	perpendicular kinematic viscosity of ions	m^2s^{-1}
ν_s^*	(Glossary) Normalised collision frequency for species s	
$\nu_c^* = \frac{q_e^4}{3m_p^2\epsilon_0^2}L_0n_0/C_0^4$	Collisionality parameter	
ν_s	(Glossary) Collision frequency for species s	s^{-1}

ν_{sn}	Collision frequency for species s with neutrals	s^{-1}
n^Z	number density for charge state Z	m^{-3}
N_Z	number of charge states for an ion species	
n_i^Z	number density for charge state Z, excited state i	m^{-3}
n_σ^Z	number density for charge state Z, metastable state σ	m^{-3}
$\omega_{ce} = \lvert e \rvert B/m_e$	electron cyclotron angular frequency	$radianss^{-1}$
$\omega_{ci} = Z_i eB/m_i$	ion cyclotron angular frequency	$radianss^{-1}$
$\omega_{pe} = \sqrt{\frac{nq_e^2}{\epsilon_0 m_e}}$	plasma angular frequency for electrons	$radianss^{-1}$
$\omega_{pi} = Z_i\sqrt{\frac{nq_e^2}{\epsilon_0 m_i}}$	plasma angular frequency for ions	$radianss^{-1}$
Ω	(K+S) Solution domain	
Ω^e	(K+S) Elemental region	
p	(K+S) Pressure	
p	pressure	Nm^{-2}
$p(A\|B)$	conditional probability of event A given event B is known or assumed to have occurred	
p_α	pressure of species α	Nm^{-2}
$\parallel Q \parallel_E$	the 'energy' norm	
$(\partial f/\partial t)_C$	source in Boltzmann due to inter-particle interactions	$m^{-6}s^2$
$\partial\Omega_e$	(K+S) Boundary of Ω^e	
$\partial\Omega$	(K+S) Boundary of Ω	
$\partial\Omega_\mathcal{D}$	(K+S) Domain boundary with Dirichlet conditions	
$\partial\Omega_\mathcal{N}$	(K+S) Domain boundary with Neumann conditions	
P_C	number of modes in basis for polynomial chaos	
p_e	pressure of the electrons	Nm^{-2}
ϕ	angle in toroidal direction	radians [c]
Φ	electric potential	V
ϕ_{pq}, ϕ_{pqr}	(FE Basis) Expansion basis	
p_i	pressure of the ion species	Nm^{-2}
P_i	(FE Basis) Polynomial order in the ith direction	
$p(\psi)$	function giving the pressure as a function of ψ of the magnetic flux	Nm^{-2}
p, q, r	(K+S) General summation indices	
Pr	Prandtl number	
Pr_M	magnetic Prandtl number	
ψ	poloidal magnetic flux	Tm^2
$\psi_p^a, \psi_{pq}^b, \psi_{pqr}^c$	(FE Basis) Modified principal functions	
Ψ_i	i^{th} member of a set of basis functions, typically multi-dimensional Hermite polynomials	
$P(T)$	emitted power integrated over all wavelengths	Wm^3
$p(x)$	probability distributions	
$P(x)$	Cumulant probability distribution	
P^Z	radiated power per atom of n^Z	W
$Q_\parallel$	combined energy flux at a boundary	$Jm^{-2}s^{-1}$

q_α	charge on a particle of species α	C
q_e	charge on an electron	C
$Q(f_\alpha, f_\beta)$	Boltzmann collision operator	$m^{-6}s^2$
Q_H	cooling rate due to excitation as defined in ref **Ha13Benc**	$Km^{-3}s^{-1}$
q_i	charge on an ion	C
q_i	electron energy flux	$Jm^{-2}s^{-1}$
q_i	ion energy flux	$Jm^{-2}s^{-1}$
Q_i	(FE Basis) Quadrature order in the ith direction	
Q_{ie}	collisional energy equipartition term	$kgm^{-1}s^{-3}$
r	order of higher order term	
R	cylindrical coordinate	m
R	recycling coefficient	
ρ	as suffix is label of metastable state	
ρ	(K+S) Density	
ρ_m	mass density of the medium	kgm^{-3}
ρ_{ts}	(Glossary) Gyroradius or Larmor radius of orbit of charged particle of species s about magnetic field direction	m
${}^R\mathcal{F}_{i\sigma}$	coefficient of recombination for the transition from metastable state σ to regular excited state i	
$s_\parallel$	arclength along fieldline	m
s	as suffix, species label	
s	parameterises distance along the fieldline $0 \leq s \leq 1$	
S_α	source term in Boltzmann equation for species α	$m^{-6}s^2$
S_C	total source term in Boltzmann equation	$m^{-6}s^2$
$S_{exp}(\mathbf{x}, \mathbf{v}, t)$	explicit source term in Boltzmann equation	$m^{-6}s^2$
n	neutral density	
T	neutral temperature	
u	neutral velocity	
s_i	arclength parameter for boundary ($i = 1$ inner, $i = 2$ outer)	
S_i	Sobol sensitivity index, gives a normalised measure of the sensitivity of the distribution of f to the parameter x_i	
σ	as suffix labels metastable state	
σ	reaction cross-section	m^2
σ_C	reaction rate for charge exchange	
σ_E	cooling rate due to excitation	
σ_E	electrical conductivity	$\Omega^{-1}m^{-1}$
σ_I	reaction rate for ionisation	
$\sigma_s^{i\|0}$	collision cross-section for ions with neutrals	m^2
$\sigma_s^{e\|0}$	collision cross-section for electrons with neutrals	m^2
S_{ij}	Sobol sensitivity index, gives a normalised measure of the sensitivity of the distribution of f to the parameters x_i and x_j	
$S^{\mathcal{E}}$	source term in plasma energy equation	$kgm^{-1}s^{-3}$
$S_e^{\mathcal{E}}$	energy density source term for electrons	$kgm^{-1}s^{-3}$

$S_i^{\mathcal{E}}$	energy density source term for ions	$kgm^{-1}s^{-3}$
$S_n^{\mathcal{E}}$	source term in neutral energy equation	$kgm^{-1}s^{-3}$
$S_{\perp e}^{\mathcal{E}}$	energy cross-field source term for electrons	$kgm^{-1}s^{-3}$
$S_{\perp i}^{\mathcal{E}}$	energy cross-field source term for ions	$kgm^{-1}s^{-3}$
$S_{\perp n}^{\mathcal{E}}$	energy cross-field source term for neutrals	$kgm^{-1}s^{-3}$
S^n	source term in plasma density equation	$m^{-3}s^{-1}$
S_e^n	number density source term for electrons	$m^{-3}s^{-1}$
S_i^n	number density source term for ions	$m^{-3}s^{-1}$
S_n^n	source term in neutral density equation	$m^{-3}s^{-1}$
$S_{\perp n}^n$	number density cross-field source term for neutrals	$m^{-3}s^{-1}$
$S_{\perp}^n$	number density cross-field source term for plasma	$m^{-3}s^{-1}$
$S_{\perp n}$	generic cross-field source term for neutrals	?
S^u	source term in plasma momentum equation	$kgm^{-2}s^{-2}$
$\subset$	(Sets) Is a subset of	
S_n^u	source term in neutral momentum equation	$kgm^{-2}s^{-2}$
$S_{\perp n}^u$	momentum cross-field source term for neutrals	$kgm^{-2}s^{-2}$
S_ρ^Z	particle source for ion of metastable state σ (species s) with charge state Z	$m^{-3}s^{-1}$
S_s^Z	particle source for ion of species s with charge state Z	$m^{-3}s^{-1}$
t	time usually in seconds	s
T	plasma temperature	eV
t_0	characteristic timescale usually in seconds	s
T_0	initial temperature	eV
T_α	temperature of species α	eV
τ_α	collision or relaxation time of species α	s
τ_e	electron collision or relaxation time	s
τ_i	ion species collision or relaxation time	s
$\tau_{e\mathsf{n}}$	electron-neutral collision time	s
$\tau_{i\mathsf{n}}$	ion species-neutral collision time	s
$\tau_{ce} = 1/f_{ce}$	electron cyclotron timescale	s
$\tau_{ci} = 1/f_{ci}$	ion cyclotron timescale	s
$\tau_{pe} = 1/f_{pe}$	plasma timescale for electrons	s
$\tau_{pi} = 1/f_{pi}$	plasma timescale for ions	s
$\tau_{\mathcal{E}_e}$	loss time of energy density for electrons	s
$\tau_{\mathcal{E}_i}$	loss time of energy density for ions	s
τ_{n_e}	loss time of number density for electrons	s
τ_{n_i}	loss time of number density for ions	s
$T_d = T_i + T_e$	combined temperature of the electrons and ions	eV
T_e	electron temperature	eV
T_H	the Hydrogen reionisation potential	
θ	angular coordinate	radians ^{c}
θ	random parameter $0 \leq \theta \leq 1$	
T_i	ion temperature	eV

$\tilde{b} = B/B_0$	magnetic field	
$\tilde{\psi}_p^a, \tilde{\psi}_{pq}^b, \tilde{\psi}_{pqr}^c$	(FE Basis) Orthogonal principal functions	
U	velocity component (flow) along streamline	ms^{-1}
$U_d = L_s/t_0$	speed measuring the importance of the transient term	ms^{-1}
U_A	Alfvén speed	ms^{-1}
$\underline{\boldsymbol{f}^e}$	(K+S) Concatenation of elemental vector $\boldsymbol{f}^e$	
$\underline{\boldsymbol{W}^e}$	(K+S) Block-diagonal extension of matrix $\boldsymbol{W}^e$	
$u_R = 1/R$	Radial component of Grad-Shafranov 'flow'	
V_i	variance of the distribution of f as the parameter x_i varies	
V_{ij}	variance of the distribution of f as the parameters x_i and x_j vary	
w_{jk}	weight in neural network indexed by neuron j and input k	
x	Cartesian coordinate	m
$x_1, x_2, x_3, \mathbf{x}$	(FE Basis) Global Cartesian coordinates	
x_α	collisionality factor of species α	
$x_e = \omega_{ce}\tau_e$	collisionality factor of electrons	
$x_i = \omega_{ci}\tau_i$	collisionality factor of ions	
x_i	generic parameter or variable	
$\xi_1, \xi_2, \xi_3, \boldsymbol{\xi}$	(FE Basis) Local Cartesian coordinates	
ξ_i	random number within the unit interval $[0, 1]$	
${}^X\mathcal{F}_{i\sigma}$	coefficient of excitation for the transition from metastable state σ to regular excited state i	
y	Cartesian coordinate	m
z	Cartesian coordinate	m
Z	Cartesian coordinate	m
Z	charge state of the ion	
Z	cylindrical coordinate	m
$Z_0(s)$	number of charge states of species s included in the model	
Z_a	Gaussian random process, index a	
ζ	magnetic Prandtl number as defined in Cambridge	
$\zeta = -\phi$	toroidal angle coordinate	radians [c]
Z_i	charge state	
$Z_m = Z - 1$	where Z is ion charge state	
$Z_p = Z + 1$	where Z is ion charge state	
$Z_{sum} = \sum_s Z_0(s)$	where Z_0 is number of charge states of species s	

Chapter 11

Index

Acknowledgement

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