~~Automating~~ Towards Automatic Finite Element Methods for Geodynamics via Firedrake

D. Rhodri Davies[1], Stephan C. Kramer[2], Sia Ghelichkhan[1], and Angus Gibson[1]

[1]Research School of Earth Sciences, The Australian National University, Canberra, ACT, Australia.
[2]Department of Earth Science and Engineering, Imperial College London, London, UK.

Correspondence: Rhodri Davies (Rhodri.Davies@anu.edu.au)

Abstract. Firedrake is an automated system for solving partial differential equations using the finite element method. By applying sophisticated performance optimisations through automatic code-generation techniques, it provides a means to create accurate, efficient, flexible, easily extensible, scalable, transparent and reproducible research software, that is ideally suited to simulating a wide-range of problems in geophysical fluid dynamics. Here, we demonstrate the applicability of Firedrake for geodynamical simulation, with a focus on mantle dynamics. The accuracy and efficiency of the approach is confirmed via comparisons against a suite of analytical and benchmark cases of systematically increasing complexity, whilst parallel scalability is demonstrated up to 12288 compute cores, where the problem size and the number of processing cores are simultaneously increased. In addition, Firedrake's flexibility is highlighted via straightforward application to different physical (e.g. complex nonlinear rheologies, compressibility) and geometrical (2-D and 3-D Cartesian and spherical domains) scenarios. Finally, a representative simulation of global mantle convection is examined, which incorporates 230 Myr of plate motion history as a kinematic surface boundary condition, confirming ~~its~~ Firedrake's suitability for addressing research problems at the frontiers of global mantle dynamics research.

1 Introduction

Since the advent of plate tectonic theory, there has been a long and successful history of research software development within the geodynamics community. The earliest modelling tools provided fundamental new insight into the process of mantle convection, its sensitivity to variations in viscosity, and its role in controlling Earth's surface plate motions and heat transport (e.g. **????**). Although transformative at the time, computational and algorithmic limitations dictated that these tools were restricted to a simplified approximation of the underlying physics and, excluding some notable exceptions (e.g. **??**), to 2-D Cartesian geometries. They were specifically designed to address targeted scientific questions~~and, accordingly,~~ . As such, they offered limited flexibility, were not easily extensible, and were not portable across different platforms. Furthermore, since they were often developed for use by one or two expert practitioners, they were poorly documented: details of the implementation could only be determined by analysing the underlying code, which was often a non-trivial and specialised task.

Growing computational resources and significant theoretical and algorithmic advances have since underpinned the development of more advanced research software, which incorporate, for example, better approximations to the fundamental phys-

ical principles, including compressibility (e.g. **?????**), mineralogical phase transformations ~~(e.g. **??**)~~(e.g. **???**), multi-phase flow (e.g. **??**), variable and nonlinear rheologies ~~(e.g. **??????????**)~~(e.g. **??????????**), and feedbacks between chemical heterogeneity and buoyancy (e.g. **???**). In addition, these more recent tools can often be applied in more representative 2-D cylindrical and/or 3-D spherical shell geometries (e.g. **??????????**). The user-base of these tools has rapidly increased, with software development teams emerging to enhance their applicability and ensure their ongoing functionality. These teams have ~~exploited~~ done so by adopting best-practices in modern software development~~best-practices~~, including version control, unit and regression testing across a range of platforms, and validation of model predictions against a suite of analytical and benchmark solutions ~~(e.g. **?????**). This has been facilitated, in part, by the expansion of community driven efforts for software development, such as the Computational Infrastructure for Geodynamics (CIG: https://www.geodynamics.org) and the Simulation and Modelling (SAM) arm of AuScope (https://www.auscope.org.au/sam).~~ (e.g. **?????**).

Nonetheless, given rapid and ongoing improvements in algorithmic design and software engineering, alongside the development of robust and flexible scientific computing libraries that provide access to much of the low-level numerical functionality required by geodynamical models, a next-generation of open-source and community driven geodynamical research software has emerged, exploiting developments from the forefront of computational engineering. This includes ASPECT (e.g. **???**), built on the deal.II (**?**), p4est (**?**) and Trilinos (**??**) libraries, Fluidity (e.g. **????**), ~~built on~~ which is underpinned by the PETSc (**???**) and Spud libraries (**?**), Underworld2 (e.g. **??**), core aspects of which are built on the St Germain (**?**) and PETSc libraries, and TerraFERMA (**?**), ~~built on~~ which has foundations in the FEniCS (**??**), PETSc and Spud libraries. By building on existing computational libraries that are highly-efficient, extensively tested and validated, modern geodynamical research software is becoming increasingly reliable and reproducible. Its modular design also ~~allows new features to be added with some~~ facilitates the addition of new features and provides a degree of confidence about the validity of previous developments, as evidenced by growth in the use and applicability of ASPECT over recent years.

However, even with these modern research software frameworks, some fundamental development decisions, such as the core physical equations, numerical approximations and general solution strategy, have been integrated into the basic building blocks of the code. Whilst there remains some flexibility within the context of a single problem, modifications to include different physical approximations or components, which can affect nonlinear coupling and associated solution strategies, often require extensive and time-consuming development and testing, using either separate code forks or increasingly complex options systems~~(**?**). This can make basic~~ . This makes reproducibility of a given ~~run difficultand results~~ simulation difficult, resulting in a lack of transparency – even with detailed documentation, specific details of the implementation are sometimes only available by reading the code itself, which, as noted previously, is non-trivial, particularly across different forks or with increasing code complexity (**?**). This makes scientific studies into the influence of different physical or geometrical scenarios, using a consistent code-base, extremely challenging. Those software frameworks that try to maintain some degree of flexibility often do so at the compromise of performance: the flexibility to configure different equations, numerical discretisations and solver strategies, in different dimensions and geometries, requires implementation compromises in the choice of optimal algorithms and specific low-level optimisations for all possible configurations.

A challenge that remains central to research software development in geodynamics, therefore, is the need to provide accurate, efficient, flexible, easily extensible, scalable, transparent and reproducible research software that can be applied to simulating a wide-range of scenarios, including problems in different geometries and those incorporating different approximations of the underlying physics (e.g. **?**). However, this requires a large time ~~investment~~ commitment and knowledge that spans several academic disciplines. Arriving at a physical description of a complex system, such as global mantle convection, demands ~~acute awareness of domain sciences:~~ expertise in geology, geophysics, geochemistry, fluid mechanics and rheology. Discretising the governing partial differential equations (PDEs) to produce a suitable numerical scheme ~~, requires expertise~~ requires proficiency in mathematical analysis, whilst its translation into efficient code for massively parallel systems demands advanced knowledge in low-level code optimisation and computer architectures (e.g. **?**). The consequence of this is that the development of research software for geodynamics has now become a multi-disciplinary effort and its design must enable scientists across several disciplines to collaborate effectively, without requiring each of them to ~~understand every aspect~~ comprehend all aspects of the system~~in detail~~.

Key to achieving this is to abstract, automate, and compose the various processes involved in numerically solving the PDEs governing a specific problem (e.g. **????**), to enable a separation of concerns between developing a technique and ~~employing~~ using it. As such, software projects involving automatic code generation have become increasingly popular, as these help to separate different aspects of development. Such an approach ~~allows for agile~~ facilitates collaboration between computational engineers with expertise in hardware and software, computer scientists and applied mathematicians with expertise in numerical algorithms, and domain specific scientists, such as geodynamicists.

In this study, we introduce Firedrake ~~(e.g. **??**)~~ to the geodynamical modelling community: a next-generation automated system for solving PDEs using the finite element method (e.g. **??**). As we will show, the finite element method is ~~highly amenable~~ well-suited to automatic code-generation techniques: a weak formulation of the ~~relevant~~ governing PDEs, together with a mesh, ~~boundary conditions~~ initial and boundary conditions, and appropriate discrete function spaces, is sufficient to ~~characterise the problemcompletely~~fully represent the problem. The purpose of this manuscript is to demonstrate the applicability of Firedrake for geodynamical simulation, whilst also highlighting its advantages over existing geodynamical research software. We do so via comparisons against a suite of analytical and benchmark cases of systematically increasing complexity.

The remainder of the manuscript is structured as follows. In Section 2, we provide a background to the Firedrake project and the various dependencies of its software stack. In Section 3 we introduce the equations governing mantle convection which will be central to the examples developed herein, followed, in Section 4, by a description of their discretisation via the finite element method and the associated solution strategies. In Section 5 we introduce a series of benchmark cases in Cartesian and spherical shell geometries. These are commonly examined within the geodynamical modelling community, and we describe the steps involved with setting up these cases in Firedrake, allowing us to highlight its ease of use. Parallel performance is analysed in Section **??**, with a representative example of global mantle convection described and analysed in Section **??**. The latter case confirms Firedrake's suitability for addressing research problems at the frontiers of global mantle dynamics research. Other components of Firedrake, which have not been showcased in this manuscript but may be beneficial to various future research endeavours, are discussed in Section **??**.

2 Firedrake

The Firedrake project is an automated system for solving partial differential equations using the finite element method (e.g. **?**). ~~'Automated', in this context, means that the user specifies the finite element problem symbolically using~~ Using a high-level language that reflects the mathematical description of the governing equations (e.g. **?**), the user specifies the finite element problem symbolically. The high-performance implementation of ~~the~~ assembly operations for the discrete operators is then generated 'automatically' by a sequence of specialised compiler passes that apply symbolic mathematical transformations to the input equations to ultimately produce C (and C++) code ~~(**?**)~~(**??**). Firedrake compiles and executes this code to create linear or nonlinear systems, which are solved by PETSc (**???**). As ~~noted~~ stated by **?**, in comparison with conventional finite element libraries, and even more so with handwritten code, Firedrake provides a higher productivity mechanism for solving finite element problems whilst simultaneously applying sophisticated performance optimisations that few users would have the resources to code by hand.

Firedrake builds on the concepts and some of the code of the FEniCS project (e.g. **?**), particularly its representation of variational problems via the Unified Form Language (UFL) (**?**). We note that the applicability of FEniCS for geodynamical problems has already been demonstrated (e.g. **??**). Both frameworks have the goal of saving users from manually writing low-level code for assembling the systems of equations that discretise their model physics. An important architectural difference is that while FEniCS has components written in C++ and Python, Firedrake is completely written in Python, including its run-time environment (it is only the automatically generated assembly code that is in C/C++, although it does leverage the PETSc library, written in C, to solve the assembled systems, albeit through its Python interface – `petsc4py`). This provides a highly flexible user interface with ease of introspection of data structures. We note that the Python environment also allows deploying of hand written C kernels should the need arise to perform discrete mesh-based operations that cannot be expressed in the finite element framework, such as sophisticated slope limiters or bespoke sub-grid physics.

Firedrake offers several highly-desirable features rendering it well-suited to problems in geophysical fluid dynamics. As will be illustrated through a series of examples below, of particular importance in the context of this manuscript are Firedrake's support for a range of different finite-element discretisations, including a highly efficient implementation of those based on extruded meshes, programmable nonlinear solvers and composable operator aware solver preconditioners. As the importance of reproducibility in the computational geosciences is increasingly recognized, we note that Firedrake integrates with Zenodo and GitHub to provide users with the ability to generate a set of DOIs corresponding to the exact set of Firedrake components used to conduct a particular simulation, in full compliance with FAIR (Findable, Accessible, Interoperable, Reusable) principles.

2.1 Dependencies

Firedrake treats finite element problems as a composition of several abstract processes, using separate packages for each. The framework imposes a clear separation of concerns between the definition of the problem (UFL, Firedrake ~~Language~~language), the generation of computational kernels used to assemble the coefficients of the discrete equations (TSFC, FInAT), the parallel execution of this kernel (PyOP2) over a given mesh topology (DMPlex), and the solution of the resulting linear or nonlinear

systems (PETSc). These layers allow various types of optimisation to be applied at different stages of the solution process. The key components of this software stack are next described.

1. Unified Form Language (UFL) – as we will see in the examples below, a core part of finite element problems is the specification of the weak form of the governing PDEs. UFL, a domain-specific symbolic language with well-defined and mathematically consistent semantics that is embedded in Python, provides an elegant solution to this problem. It was pioneered by the FEniCS project (**?**), although Firedrake has added several extensions.

2. Firedrake ~~Language~~ language – in addition to the weak form of the PDEs, finite element problems require the user to select appropriate finite elements, specify the mesh to be employed, set field values for initial and boundary conditions and specify the sequence in which solves occur. Firedrake implements its own language for these tasks, which was designed to be to a large extent compatible with DOLFIN (**?**), the runtime API of the FEniCS project. We note that Firedrake implements various extensions to DOLFIN, whilst some features of DOLFIN are not supported by Firedrake.

3. FInAT (**?**) – incorporates all information required to evaluate the basis functions of the different finite element families supported by Firedrake. In earlier versions of Firedrake this was done through tabulation of the basis functions evaluated at Gauss points (FIAT: **?**). FInAT, however, provides this information to the form compiler as a combination of symbolic expressions and numerical values, allowing for further optimisations. FInAT allows Firedrake to support a wide-range of finite elements, including continuous, discontinuous, H(div) and H(curl) discretisations, and elements with continuous derivatives such as the Argyris and Bell elements.

4. Two-Stage Form Compiler (TSFC) – a form compiler takes a high-level description of the weak form of PDEs (here in UFL) and produces low-level code that carries out the finite element assembly. Firedrake uses TSFC, which was developed specifically for the Firedrake project (**?**), to generate its local assembly kernels. TSFC invokes two stages, where in the first stage UFL is translated to an intermediate symbolic tensor algebra language, before translating this into assembly kernels written in C. In comparison with the form compilers of FEniCS (FFC and UFLACS), TSFC aims to maintain the algebraic structure of the input expression for longer, which opens up additional opportunities for optimisation.

5. PyOP2 – a key component of Firedrake's software stack is PyOP2, a high-level framework that optimises the parallel execution of computational kernels on unstructured meshes (**??**). Where the local assembly kernels generated by TSFC calculate the values of a local tensor from local input tensors, all associated with the degrees of freedom of a single element, PyOP2 wraps this code in an additional layer responsible for the extraction and addition of these local tensors out of/into global structures such as vectors and sparse matrices. It is also responsible for the maintenance of halo layers, the overlapping regions in a parallel decomposed problem. PyOP2 allows for a clean separation of concerns between the specification of the local kernel functions, in which the numerics of the method are encoded, and their efficient parallel execution. More generally, this separation of concerns is the key novel abstraction that underlies the design of the Firedrake system.

6. DMPlex – PyOP2 has no concept of the topological construction of a mesh. Firedrake derives the required maps through DMPlex, a data management abstraction that represents unstructured mesh data, which is part of the PETSc project (**?**). This allows Firedrake to leverage the DMPlex partitioning and data migration interfaces to perform domain decomposition at run-time, whilst supporting multiple mesh file formats. Moreover, Firedrake reorders mesh entities to ensure computational efficiency (**?**).

7. Linear and nonlinear solvers – Firedrake passes solver problems on to ~~the established,~~ PETSc (**???**), a well-established, high-performance solver library ~~, PETSc (**???**), which~~ that provides access to several of its own and third-party implementations of solver algorithms. The Python interface to PETSc (**?**) makes ~~its~~ integration with Firedrake straightforward. We note that employing PETSc for both its solver library and for DMPlex has the additional advantage that the set of library dependencies required by Firedrake is kept small (**?**).

3 Governing Equations

Our focus here is on mantle convection, the slow creeping motion of Earth's mantle over geological timescales. The equations governing mantle convection are derived from the conservation laws of mass, momentum and energy. The simplest mathematical formulation assumes ~~incompressibility~~ a single incompressible material and the Boussinesq approximation (**?**), under which the non–dimensional momentum and continuity equations are given by:

$$\nabla \cdot \bar{\bar{\sigma}} + Ra_0 T \hat{\boldsymbol{k}} = 0, \tag{1}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{2}$$

where $\bar{\bar{\sigma}}$ is the stress tensor, $\boldsymbol{u}$ is the velocity and T temperature. $\hat{\boldsymbol{k}}$ is the unit vector in the direction opposite to gravity and Ra_0 denotes the Rayleigh number, a dimensionless number that quantifies the vigor of convection:

$$Ra_0 = \frac{\rho_0 \alpha \Delta T g d^3}{\mu_0 \kappa}. \tag{3}$$

Here, ρ_0 denotes reference density, α the thermal expansion coefficient, ΔT the characteristic temperature change across the domain, g the gravitational acceleration, d the characteristic length, μ_0 the reference dynamic viscosity and κ the thermal diffusivity. Note that the above non–dimensional equations are obtained through the following characteristic scales: length d; time d^2 / κ; and temperature ΔT.

When simulating incompressible flow, the full stress tensor, $\bar{\bar{\sigma}}$, is decomposed into deviatoric and volumetric components:

$$\bar{\bar{\sigma}} = \bar{\bar{\tau}} - pI, \tag{4}$$

where $\bar{\bar{\tau}}$ is the deviatoric stress tensor, p is dynamic pressure and I is the identity matrix. Substituting Eq. (4) into Eq. (1) and utilizing the ~~constituative~~ constitutive relation

$$\bar{\bar{\tau}} = 2\mu\dot{\epsilon} = 2\mu \,\mathrm{sym}(\nabla \boldsymbol{u}) = \mu \left[\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T \right], \tag{5}$$

which relates the deviatoric stress tensor, $\bar{\bar{\tau}}$, to the strain-rate tensor, $\dot{\epsilon} = \mathrm{sym}(\nabla \boldsymbol{u})$, yields:

$$\nabla \cdot \mu \left[\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T \right] - \nabla p + Ra_0 T \hat{\boldsymbol{k}} = 0. \tag{6}$$

The viscous flow problem can thus be posed in terms of pressure, p, velocity, $\boldsymbol{u}$, and temperature, T. The evolution of the thermal field is controlled by an advection–diffusion equation:

$$\frac{\partial T}{\partial t} + \boldsymbol{u} \cdot \nabla T - \nabla \cdot (\kappa \nabla T) = 0 \tag{7}$$

These governing equations are sufficient to solve for the three unknowns, together with adequate boundary and initial conditions.

4 Finite Element Discretisation and Solution Strategy

For the derivation of the finite element discretisation of Equations (6), (2), and (7) we start by writing these in their weak form. We select appropriate function spaces V, W, and Q that contain, respectively, the solution fields for velocity $\boldsymbol{u}$, pressure p, and temperature T, and also contain the test functions $\boldsymbol{v}, w$ and q. The weak form is then obtained by multiplying these equations with the test functions and integrating over the domain Ω,

$$\int_\Omega (\nabla \boldsymbol{v}) : \mu \left[\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right] \, \mathrm{d}\boldsymbol{x} \underline{+}\underset{\sim}{-} \int_\Omega \underline{:} \left(\nabla \cdot \boldsymbol{v} \right) p \, \mathrm{d}\boldsymbol{x} - \int_\Omega Ra_0 T \boldsymbol{v} \cdot \hat{\boldsymbol{k}} \, \mathrm{d}\boldsymbol{x} = 0 \text{ for all } \boldsymbol{v} \in V, \tag{8}$$

$$\underline{-} \int_\Omega \underline{\nabla} w \underset{\sim}{\nabla} \cdot \mathbf{u} \, \mathrm{d}\boldsymbol{x} = 0 \text{ for all } w \in W, \tag{9}$$

$$\int_\Omega q \frac{\partial T}{\partial t} \, \mathrm{d}\boldsymbol{x} + \int_\Omega q \boldsymbol{u} \cdot \nabla T \, \mathrm{d}\boldsymbol{x} + \int_\Omega (\nabla q) \cdot (\kappa \nabla T) \, \mathrm{d}\boldsymbol{x} = 0 \text{ for all } q \in Q. \tag{10}$$

Note that we have integrated by parts the viscosity ~~term~~ and pressure gradient terms in (6), ~~the divergence term in ,~~ and the diffusion term in (7), but have omitted the corresponding boundary terms, which will be considered in the following section.

Equations (8-10) are a more general ~~, mathematically rigorous~~ representation of the continuous PDEs in strong form (Equations 6, 2 and 7), provided suitable function spaces with sufficient regularity are chosen (see, for example **??**). ~~Galerkin finite~~ Finite element discretisation proceeds by restricting these function spaces to finite-dimensional subspaces. These are typically constructed by dividing the domain into cells or elements, and restricting to piecewise polynomial subspaces with various continuity requirements between cells. ~~In all examples presented in~~ Firedrake offers a very wide range of such finite element function spaces (see **?**, for an overview). It should be noted however that, in practice, this choice is guided by numerical stability considerations in relation to the specific equations that are being solved. In particular, the choice of velocity and pressure function spaces used in the Stokes system is restricted by the LBB condition (see **?**, for an overview of common choices for geodynamical f

In this paper, we ~~use Continuous Galerkin (CG) finite elements, specifically the Q2-Q1~~ focus on the use of the familiar Q2Q1 element pair for velocity and pressure~~and the~~ , which employs piecewise continuous bi-quadratic and bi-linear polynomials on quadrilaterals or hexahedra for velocity and pressure, respectively. In addition, to showcase Firedrake's flexibility, we use

the less familiar, $Q2P_{1DG}$ pair in a number of cases, in which pressure is discontinuous and piecewise linear (but not bilinear). For temperature, we primarily use a Q2 ~~discretisation for temperature.~~ discretisation, but also show some results using a Q1 discretisation.

~~We note that there are many other choices of finite element function spaces available in Firedrake, although they are not considered herein (see ?, for an overview).~~ All that is required for ~~their implementation~~ the implementation of these choices is that a basis can be found for the function space such that each solution can be written as a linear combination of basis functions. For example, if we have a basis ϕ_i of the finite dimensional function space Q_h of temperature solutions, then we can write each temperature solution as

$$T(\boldsymbol{x}) = \sum_i T_i \phi_i(\boldsymbol{x}) \tag{11}$$

where T_i represents the coefficients that we can collect into a discrete solution vector $\underline{\mathrm{T}}$. Using a Lagrangian polynomial basis the coefficients T_i correspond to values at the nodes, where each node i is associated with one basis function ϕ_i, but this is not generally true for other choices of finite element bases.

In curved domains, boundaries can only be approximated with a finite number of triangles, tetrahedrals, quadrilaterals or hexahedrals. In a sense, this can be seen as a piecewise linear (or bi/tri-linear) approximation where the domain is approximated by straight lines (edges) between vertices. A more accurate representation of the domain is obtained by allowing higher order polynomials that describe the physical embedding of the element within the domain. A typical choice is to use a so-called isoparametric representation in which the polynomial order of the embedding is the same as that of the discretised functions that are solved for.

Finally, we note that it is common to use a subscript $_h$ for the discrete, finite-dimensional function subspaces and Ω_h for the discretised approximation by the mesh of the domain Ω, but since the remainder of this manuscript focusses on the details and implementation of this discretisation, we simply drop the $_h$ subscripts from here on.

4.1 Boundary conditions

In the Cartesian examples considered below, zero-slip and free-slip boundary conditions for Equations (8) and (9) are imposed through strong Dirichlet boundary conditions for velocity $\boldsymbol{u}$. This is achieved by restricting the velocity function space V to a subspace V_0 of vector functions for which all components (zero-slip) or only the normal component (free-slip) are zero at the boundary. Since this restriction also applies to the test functions $\boldsymbol{v}$, the weak form only needs to be satisfied for all test functions $\boldsymbol{v} \in V_0$ that satisfy the homogeneous boundary conditions. Therefore, the omitted boundary integral

$$-\int_{\partial\Omega} \boldsymbol{v} \cdot \left(\mu \left[\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T\right]\right) \cdot \boldsymbol{n} \, \mathrm{d}\boldsymbol{s} \tag{12}$$

that was required to obtain the integrated by parts viscosity term in Equation (8), automatically vanishes for zero-slip boundary conditions as $\mathbf{v} = \mathbf{0}$ at the domain boundary, $\partial\Omega$. In the case of a free-slip boundary condition for which the tangential components of $\boldsymbol{v}$ are non-zero, the boundary term does not vanish, but by omitting that term in Equation (8) we weakly impose a zero

shear stress condition. The boundary term obtained by integrating the ~~divergence term in~~ pressure gradient term in Equation (2) by parts,

$$\int_{\partial\Omega} w\boldsymbol{v}\cdot\boldsymbol{n}p\,\mathrm{d}\boldsymbol{s}, \tag{13}$$

~~vanishes for both~~ also vanishes as $\boldsymbol{v}\cdot\boldsymbol{n}=0$ for $\boldsymbol{v}\in V_0$ in both the zero-slip and free-slip ~~boundary conditions because of the no-outflow boundary condition~~cases.

Similarly, in the examples presented below, we impose strong Dirichlet boundary conditions for temperature at the top and bottom boundaries of our domain. The test functions are restricted to Q_0 which consists of temperature functions that satisfy homogeneous boundary conditions at these boundaries, and thus

$$\int_{\partial\Omega} q\boldsymbol{n}\cdot\kappa\nabla T\,\mathrm{d}\boldsymbol{s}, \tag{14}$$

the boundary term associated with integrating by parts of the diffusion term, vanishes. In Cartesian domains the boundary term does not vanish for the lateral boundaries, but by omitting this term from Equation (10) we weakly impose a homogeneous Neumann (zero-flux) boundary condition at these boundaries. The temperature solution itself is found in $Q_0+\{T_{\text{inhom}}\}$ where T_{inhom} is any representative temperature function that satisfies the required inhomogenous boundary conditions.

In curved domains, such as the 2-D cylindrical shell and 3-D spherical shell cases examined below, imposing free-slip boundary conditions is complicated by the fact that it is not straightforward to decompose the degrees of freedom of the velocity space V into tangential and lateral components for many finite element discretisations. For Lagrangian based discretisations we could define normal vectors at the Lagrangian nodes on the surface and decompose accordingly, but these normal vectors would have to be averaged due to the piecewise approximation of the curved surface. To avoid such complications for our examples in cylindrical and spherical geometries, we employ a symmetric Nitsche penalty method (**?**) where the velocity space is not restricted and, thus, retains all discrete solutions with a non-zero normal component. This entails adding the following three surface integrals to Equation (8):

$$-\int_{\partial\Omega}\boldsymbol{v}\cdot\boldsymbol{n}\,\boldsymbol{n}\cdot\left(\mu\left[\nabla\boldsymbol{u}+(\nabla\boldsymbol{u})^T\right]\right)\cdot\boldsymbol{n}\,\mathrm{d}\boldsymbol{s}-\int_{\partial\Omega}\boldsymbol{n}\cdot\left(\mu\left[\nabla\boldsymbol{v}+(\nabla\boldsymbol{v})^T\right]\right)\cdot\boldsymbol{n}\,\boldsymbol{u}\cdot\boldsymbol{n}\,\mathrm{d}\boldsymbol{s}+\int_{\partial\Omega}C_{\text{Nitsche}}\,\mu\boldsymbol{v}\cdot\boldsymbol{n}\,\boldsymbol{u}\cdot\boldsymbol{n}\,\mathrm{d}\boldsymbol{s}\quad. \tag{15}$$

The first of these corresponds to the normal component of Equation (12) associated with integration by parts of the viscosity term. The tangential component, as before, is omitted and weakly imposes a zero shear stress condition. The second term ensures symmetry of Equation (8) with respect to $\boldsymbol{u}$ and $\boldsymbol{v}$. The third term penalizes the normal component of $\boldsymbol{u}$ and involves a penalty parameter $C_{\text{Nitsche}}>0$ that should be sufficiently large to ensure coercivity of the bilinear form F_{Stokes} ~~as a bilinear form in $\boldsymbol{u}$ and $\boldsymbol{v}$~~introduced in Section 4.3. Lower bounds for $C_{\text{Nitsche},f}$ on each face f can be derived for simplicial (**?**) and quadrilateral/hexahedral (**?**) meshes, respectively:

Triangular ($d=2$) / Tetrahedral ($d=3$) meshes:
$$C_{\text{Nitsche},f}>C_{\text{ip}}\frac{p(p+d-1)}{d}\frac{A_f}{V_{c_f}}, \tag{16}$$

Quadrilateral/Hexahedral meshes:
$$C_{\text{Nitsche},f}>C_{\text{ip}}(p+1)^2\frac{A_f}{V_{c_f}}, \tag{17}$$

where A_f is the facet area of face f, V_{c_f} the cell volume of the adjacent cell c_f, and p is the polynomial degree of the velocity discretisation. Here, we introduce an additional factor, C_{ip}, to account for spatial variance of the viscosity μ in the adjacent cell, and domain curvature, which are not taken into account in the standard lower bounds (using $C_{\text{ip}} = 1$). In all free-slip cylindrical and spherical shell examples presented below, we use $C_{\text{ip}} = 100$. Finally, because the normal component of velocity is not restricted in the velocity function space, the boundary term (13) no longer vanishes, and we also need to weakly impose the no-normal flow condition on the continuity equation by adding the following integral to Equation (9):

$$-\int_{\partial\Omega} w\boldsymbol{n}\cdot\boldsymbol{u}\,\mathrm{d}\boldsymbol{s}. \tag{18}$$

4.2 Temporal discretisation and solution process for temperature

For temporal integration, we apply a simple θ scheme to the energy equation (10):

$$F_{\text{energy}}(q;T^{n+1}) := \int_\Omega q\frac{T^{n+1}-T^n}{\Delta t}\,\mathrm{d}\boldsymbol{x} + \int_\Omega q\boldsymbol{u}\cdot\nabla T^{n+\theta}\,\mathrm{d}\boldsymbol{x} + \int_\Omega (\nabla q)\cdot\left(\kappa\nabla T^{n+\theta}\right)\,\mathrm{d}\boldsymbol{x} = 0 \text{ for all } q\in Q, \tag{19}$$

where

$$T^{n+\theta} = \theta T^{n+1} + (1-\theta)T^n \tag{20}$$

is interpolated between the temperature solutions T^n and T^{n+1} at the beginning and end of the $n+1$-th time step using a parameter $0 \le \theta \le 1$. In all examples that follow, we use a ~~Crank-Nicholson~~ Crank-Nicolson scheme, where $\theta = 0.5$. ~~To simplify we will solve~~ It should be noted that the time-dependent energy equation is coupled with the Stokes system through the buoyancy term and, in some cases, the temperature-dependence of viscosity and. At the same time, the Stokes equation couples to the energy equation through the advective velocity. These combined equations can therefore be considered as a coupled system that should be iterated over. The solution algorithm used here follows a standard time-splitting approach. We solve the Stokes system for velocity and pressure ~~, $\boldsymbol{u}$ and p, in a separate step before solving~~ with buoyancy and viscosity terms, based on a given prescribed initial temperature field. In a separate step, we solve for the new temperature T^{n+1} ~~.~~ using the new velocity, advance in time and repeat. The same time loop is used to converge the coupling in steady-state cases.

Because F_{energy} is linear in q, if we expand the test function q as a linear combination of basis functions ϕ_i of Q

$$F_{\text{energy}}(q;T^{n+1}) = F_{\text{energy}}(\sum_i q_i\phi_i;T^{n+1}) = \sum_i q_i F_{\text{energy}}(\phi_i;T^{n+1}) =: \sum_i q_i \underline{\mathrm{F}}(T^{n+1})_i, \tag{21}$$

where $\underline{\mathrm{F}}(T^{n+1})$ is the vector with coefficients $F_{\text{energy}}(\phi_i;T^{n+1})$ (i.e. the energy equation tested with the basis functions ϕ_i). Thus, to satisfy Equation (19) we need to solve for a temperature T for which the entire vector $\underline{\mathrm{F}}(T^{n+1})$ is zero.

In the general ~~, nonlinear case ,~~ nonlinear case (for example, if the thermal diffusivity is temperature dependent), this can be solved using a Newton solver, but here the system of equations $\underline{\mathrm{F}}(T^{n+1})$ is also linear in T^{n+1} and, accordingly, if we also expand the temperature with respect to the same basis: $T^{n+1} = \sum_j T_j^{n+1}\phi_j$ where we store the coefficients T_j^{n+1} in a vector

$\underline{\mathrm{T}}$, we can write it in the usual form as a linear system of equations

$$A\underline{\mathrm{T}} = \underline{\mathrm{b}}, \tag{22}$$

with A the matrix that represents the Jacobian $\frac{\partial F}{\partial T}$ with respect to the basis ϕ_i, and the right-hand side vector $\underline{\mathrm{b}}$ containing all terms in (19) that do not depend on T^{n+1}, specifically:

$$A_{ij} = \frac{\partial F_{\text{energy}}(\phi_i; T^{n+1})}{\partial T_j^{n+1}} = \int_\Omega \phi_i \frac{\phi_j}{\Delta t}\,\mathrm{d}\boldsymbol{x} + \int_\Omega \phi_i \boldsymbol{u}\cdot\theta\nabla\phi_j\,\mathrm{d}\boldsymbol{x} + \int_\Omega (\nabla\phi_i)\cdot(\kappa\theta\nabla\phi_j)\,\mathrm{d}\boldsymbol{x} = 0 \tag{23}$$

$$\underline{\mathrm{b}}_j = F - F_{\text{energy}}(\phi_i; 0) = \int_\Omega \phi_i \frac{T^n}{\Delta t}\,\mathrm{d}\boldsymbol{x} - \int_\Omega \phi_i \boldsymbol{u}\cdot(1-\theta)\nabla T^n\,\mathrm{d}\boldsymbol{x} - \int_\Omega (\nabla\phi_i)\cdot(\kappa(1-\theta)\nabla T^n)\,\mathrm{d}\boldsymbol{x} = 0 \tag{24}$$

In the nonlinear case, every Newton iteration requires the solution of such a linear system with a Jacobian matrix $A_{ij} = \partial F_{\text{energy}}/\partial T_j^{n+1}$ and a right-hand side vector based on the residual $\underline{\mathrm{b}}_i = F_{\text{energy}}(\phi_i, T^{n+1})$ ~~that both need~~ both of which are to be reassembled every iteration as T^{n+1} is iteratively improved. For the 2-D cases presented in this paper, this asymmetric linear system is solved with a direct solver, and in 3-D using a combination of the GMRES Krylov subspace method with a symmetric SOR (SSOR) preconditioner.

4.3 Solving for velocity and pressure

In a separate step, we solve Equations (8) and (9) for velocity and pressure. Since these weak equations need to hold for all test functions $\boldsymbol{v} \in V$ and $w \in W$ we can equivalently write, using a single residual functional F_{Stokes}:

$$\begin{aligned} F_{\text{Stokes}}(\boldsymbol{v}, w; \boldsymbol{u}, p) = \int_\Omega (\nabla\boldsymbol{v}) : \mu\left[\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right]\,\mathrm{d}\boldsymbol{x} - \int_\Omega \left(\nabla\cdot\boldsymbol{v}\right) p\,\mathrm{d}\boldsymbol{x} \\ - \int_\Omega Ra_0 T\boldsymbol{v}\cdot\hat{\boldsymbol{k}}\,\mathrm{d}\boldsymbol{x} - \int_\Omega \nabla w \nabla\cdot\boldsymbol{u}\,\mathrm{d}\boldsymbol{x} = 0 \text{ for all } \boldsymbol{v}\in V, w\in W, \end{aligned} \tag{25}$$

where we have multiplied the continuity equation with -1 to ensure symmetry between the ∇p and ~~$\nabla\cdot u$~~ $\nabla\cdot\boldsymbol{u}$ terms. This combined weak form that we simultaneously solve for a velocity ~~$u \in V$~~ $\boldsymbol{u} \in V$ and pressure $p \in W$ is referred to as a *mixed problem*, and the combined solution ~~(u,p)~~ $(\boldsymbol{u},p)$ is said to be found in the *mixed function space* $V \oplus W$.

As before, we expand the discrete solutions $\boldsymbol{u}$ and p, and test functions $\boldsymbol{v}$ and w in terms of basis functions for V and W

$$\boldsymbol{u} = \sum_i u_i \boldsymbol{\psi}_i, \qquad \boldsymbol{v} = \sum_i v_i \boldsymbol{\psi}_i, \qquad \operatorname{span}\{\boldsymbol{\psi}_i\} = V \tag{26}$$

$$p = \sum_k p_k \chi_k, \qquad w = \sum_k w_k \chi_k, \qquad \operatorname{span}\{\chi_k\} = W \tag{27}$$

For isoviscous cases, where F_{Stokes} is linear in $\boldsymbol{u}$ and p, we then derive a linear system of the following form

$$\begin{pmatrix} K & G \\ G^T & 0 \end{pmatrix} \begin{pmatrix} \underline{\mathrm{u}} \\ \underline{\mathrm{p}} \end{pmatrix} = \begin{pmatrix} \underline{\mathrm{f}} \\ \underline{0} \end{pmatrix} \tag{28}$$

where

$$K_{ij} = \frac{\partial F_{\text{Stokes}}(\boldsymbol{\psi}_i, 0; \boldsymbol{u}, p)}{\partial \boldsymbol{u}_j} \frac{\partial F_{\text{Stokes}}(\boldsymbol{\psi}_i, 0; \boldsymbol{u}, p)}{\partial u_j} = \int_\Omega (\nabla \boldsymbol{\psi}_i) : \mu \left[\nabla \boldsymbol{\psi}_j + (\nabla \boldsymbol{\psi}_j)^T \right] \mathrm{d}\boldsymbol{x} \tag{29}$$

$$G_{ik} = \frac{\partial F_{\text{Stokes}}(\boldsymbol{\psi}_i, 0; \boldsymbol{u}, p)}{\partial p_k} = -\int_\Omega \left(\nabla \cdot \boldsymbol{\psi}_i \right) \nabla \chi_k \, \mathrm{d}\boldsymbol{x} = -\frac{\partial F_{\text{Stokes}}(0, \chi_k; \boldsymbol{u}, p)}{\partial \boldsymbol{u}_i} \frac{\partial F_{\text{Stokes}}(0, \chi_k; \boldsymbol{u}, p)}{\partial u_i} \tag{30}$$

$$\underline{\mathrm{f}}_i = Ra_0 \int_\Omega T \boldsymbol{\psi}_i \cdot \hat{\boldsymbol{k}} \, \mathrm{d}\boldsymbol{x} \tag{31}$$

For cases with more general rheologies, in particular those with a strain-rate dependent viscosity, the system $\underline{\mathrm{F}}_{\text{Stokes}}(\boldsymbol{u}, p) = \underline{0}$ is nonlinear and can be solved using Newton's method. This requires the solution in every Newton iteration of a linear system of the same form as in Equation (28) but with an additional term in K associated with $\partial\mu/\partial\boldsymbol{u}$. For the strain-rate dependent cases presented in this paper this takes the following form

$$K_{ij} = \frac{\partial F_{\text{Stokes}}(\boldsymbol{\psi}_i, 0; \boldsymbol{u}, p)}{\partial u_j} = \int_\Omega (\nabla \boldsymbol{\psi}_i) : \mu \left[\nabla \boldsymbol{\psi}_j + (\nabla \boldsymbol{\psi}_j)^T \right] \mathrm{d}\boldsymbol{x} + \int_\Omega (\nabla \boldsymbol{\psi}_i) : (\nabla \boldsymbol{u}) \frac{\partial \mu(\dot{\epsilon})}{\partial \dot{\epsilon}} : \left[\nabla \boldsymbol{\psi}_j + (\nabla \boldsymbol{\psi}_j)^T \right] \mathrm{d}\boldsymbol{x} \tag{32}$$

Note that the additional term makes the matrix explicitly dependent on the solution $\boldsymbol{u}$ itself, and is asymmetric. Here, for brevity we have not expanded the derivative of μ with respect to the strain-rate tensor $\dot{\epsilon}$. Such additional terms require a significant amount of effort to implement in traditional codes and need adapting to the specific rheological approximation that is used, but this is all handled automatically here through the combination of symbolic differentiation and code generation in Firedrake.

There is a wide literature on iterative methods for solving saddle point systems of the form in Equation (28). For an overview of the methods commonly used in geodynamics, see **?**. Here we employ the Schur complement approach, where pressure $\underline{\mathrm{p}}$ is determined by solving

$$G^T K^{-1} G \underline{\mathrm{p}} = G^T K^{-1} \underline{\mathrm{f}} \quad . \tag{33}$$

It should be noted that K^{-1} is not assembled explicitly. Rather, in a first step we obtain $\underline{\mathrm{y}} = K^{-1}\underline{\mathrm{f}}$ by solving $K\underline{\mathrm{y}} = \underline{\mathrm{f}}$ so that we can construct the right-hand side of the equation. We subsequently apply ~~an~~ the flexible GMRES (**?**) iterative method to the linear system as a whole, in which each iteration requires matrix-vector multiplication with the matrix $G^T K^{-1} G$ that again involves the solution of a linear system with matrix K. ~~In addition to this matrix-vector multiplication we~~ We also need a suitable preconditioner. Here we follow the inverse scaled-mass matrix approach which uses the following approximation

$$G^T K^{-1} G \approx M, \quad M_{ij} = \int_\Omega \mu \boldsymbol{\psi}_i \boldsymbol{\psi}_j \tag{34}$$

Finally, after solving Equation (33) for $\underline{\mathrm{p}}$, we obtain $\underline{\mathrm{u}}$ in a final solve $K\underline{\mathrm{u}} = \underline{\mathrm{f}} - G\underline{\mathrm{p}}$.

Since this solution process involves multiple solves with the matrix K, we also need an efficient algorithm to solve that system. For this, we combine the conjugate gradient method with an algebraic multigrid approach, specifically the Geometric Algebraic Multigrid (GAMG) method implemented in PETSc (**???**).

Depending on boundary conditions, the linearised Stokes system admits a number of null modes. In the absence of open boundaries, which is the case for all cases examined here, the pressure admits a constant null mode, where any arbitrary constant can be added to the pressure solution and remain a valid solution to the equations. In addition, ~~the~~ cylindrical and spherical shell cases with free-slip boundary conditions at both boundaries ~~examined in Section 5,~~ admit, respectively, one and three independent rotational null modes in velocity. ~~These~~ As these null modes result in ~~indefinite matricesand~~ singular matrices, preconditioned iterative methods ~~typically require~~ should typically be provided with the null vectors~~to be provided so that they can be projected out during iteration~~.

In the absence of any Dirichlet conditions on velocity, the nullspace of the velocity block K also consists of a further two independent translational modes in 2D, and three in 3D. Even ~~if, as for the cases here, the~~ in simulations where boundary conditions do not admit ~~all~~ any rotational and translational modes, these solutions ~~are still~~ remain associated with low energy modes of the matrix~~, and some~~. Some multigrid methods use this information to improve their performance by ensuring that these ~~near-nullspace~~ so-called *near-nullspace* modes are accurately represented at the coarser levels (**?**). We make use of this in several of the examples considered below.

5 Examples: Benchmark Cases and Validation

Firedrake provides a complete ~~finite element problem-solving environment~~framework for solving finite element problems, highlighted in this section through a series of examples. We start in Section 5.1 with the most basic problem – isoviscous, incompressible convection, in an enclosed 2-D Cartesian box – and systematically build complexity, initially moving into more realistic physical approximations (Section 5.2) and, subsequently, geometries that are more representative of Earth's mantle (Section 5.3). The cases examined, and the challenges associated with each, are summarised in Table 1.

5.1 Basic Example: 2-D Convection in a Square Box

A simple 2-D square convection problem, from **?**, for execution in Firedrake, is displayed in Listing 1. The problem is incompressible, isoviscous, heated from below and cooled from above, with closed, free-slip boundaries, on a unit square mesh.

Name	**Source**	**Geometry**	**Rheology**	**Additional Functionality**
Base Case	**?**	2-D Cartesian	Isoviscous	–
2-D Compressible	**?**	2-D Cartesian	Isoviscous	UFL changes, reference state, boundary conditions (BCs)
2-D Viscoplastic	**?**	2-D Cartesian	$\mu(T, z, \dot{\epsilon})$	μ calculation, nonlinear solvers (SNES)
3-D Cartesian	**?**	3-D Cartesian	Isoviscous	Iterative solvers, near-nullspaces (NNS)
2-D Cylindrical Shell	–	2-D Cylindrical Shell	Isoviscous	Radial $\boldsymbol{g}$, Nitsche BCs, nullspaces, NNS, iterative solvers
3-D Spherical Shell	**?**	3-D Spherical Shell	Isoviscous	Radial $\boldsymbol{g}$, Nitsche BCs, nullspaces, NNS, iterative solvers
Global Circulation	–	3-D Spherical Shell	$\mu(T, z, \dot{\epsilon})$	Radial $\boldsymbol{g}$, BCs (Nitsche, GPlates), NNS, iterative solvers

Table 1. Summary of cases examined here, which systematically increase in complexity. The key differences and challenges differentiating each case from the base case are highlighted in the final column.

```python
from firedrake import *

# Mesh - use a built in meshing function:
mesh = UnitSquareMesh(40, 40, quadrilateral=True)
left, right, bottom, top = 1, 2, 3, 4  # Boundary IDs
n = FacetNormal(mesh)  # Normals, required for Nusselt number
domain_volume = assemble(1.*dx(domain=mesh))  # Required for RMS velocity

# Function spaces:
V = VectorFunctionSpace(mesh, family="CG", degree=2)  # Velocity function space (vector)
W = FunctionSpace(mesh, family="CG", degree=1)  # Pressure function space (scalar)
Q = FunctionSpace(mesh, family="CG", degree=2)  # Temperature function space (scalar)
Z = MixedFunctionSpace([V, W])  # Mixed function space

# Test functions and functions to hold solutions:
v, w = TestFunctions(Z)
q = TestFunction(Q)
z = Function(Z)
u, p = split(z)  # Returns symbolic UFL expression for u and p
Told, Tnew = Function(Q, name="OldTemp"), Function(Q, name="NewTemp")
Ttheta = 0.5 * Tnew + 0.5 * Told  # Temporal discretisation through Crank-Nicholson

# Initialise temperature field:
X = SpatialCoordinate(mesh)
Told.interpolate(1.0 - X[1] + 0.05 * cos(pi * X[0]) * sin(pi * X[1]))
Tnew.assign(Told)

# Important constants:
Ra, mu, kappa, delta_t = Constant(1e4), Constant(1.0), Constant(1.0), Constant(1e-6)
k = Constant((0, 1))  # Unit vector (in direction opposite to gravity)

# Stokes equations in UFL form:
stress = 2 * mu * sym(grad(u))
F_stokes = inner(grad(v), stress) * dx - div(v) * p * dx - (dot(v, k) * Ra * Ttheta) * dx
F_stokes += -w * div(u) * dx  # Continuity equation
# Energy equation in UFL form:
F_energy = q * (Tnew - Told) / delta_t * dx + q * dot(u, grad(Ttheta)) * dx + dot(grad(q), kappa * grad(Ttheta)) * dx

# Set up boundary conditions and deal with nullspaces:
bcvx, bcvy = DirichletBC(Z.sub(0).sub(0), 0, sub_domain=(left, right)), DirichletBC(Z.sub(0).sub(1), 0, sub_domain=(bottom, top))
bctb, bctt = DirichletBC(Q, 1.0, sub_domain=bottom), DirichletBC(Q, 0.0, sub_domain=top)
p_nullspace = MixedVectorSpaceBasis(Z, [Z.sub(0), VectorSpaceBasis(constant=True)])

# Initialise output:
output_file = File('output.pvd')  # Create output file
u_, p_ = z.split()
u_.rename("Velocity"), p_.rename("Pressure")

# Solver dictionary:
solver_parameters = {
    "mat_type": "aij",
    "snes_type": "ksponly",
    "ksp_type": "preonly",
    "pc_type": "lu",
    "pc_factor_mat_solver_type": "mumps"}

# Setup problem and solver objects so we can reuse (cache) solver setup
stokes_problem = NonlinearVariationalProblem(F_stokes, z, bcs=[bcvx, bcvy])
stokes_solver = NonlinearVariationalSolver(stokes_problem, solver_parameters=solver_parameters, nullspace=p_nullspace, transpose_nullspace=p_nullspace)
energy_problem = NonlinearVariationalProblem(F_energy, Tnew, bcs=[bctb, bctt])
energy_solver = NonlinearVariationalSolver(energy_problem, solver_parameters=solver_parameters)

# Timestepping aspects
no_timesteps, target_cfl_no = 2000, 1.0
ref_u = Function(V, name="Reference_Velocity")


def compute_timestep(u):
    """Return the timestep, using CFL criterion"""
    tstep = (1. / ref_u.interpolate(dot(JacobianInverse(mesh), u)).dat.data.max()) * target_cfl_no
    return tstep


for timestep in range(0, no_timesteps):
    if timestep > 0:
        delta_t.assign(compute_timestep(u))
    if timestep % 10 == 0:
        output_file.write(u_, p_, Tnew)
    stokes_solver.solve()
    energy_solver.solve()
    vrms = sqrt(assemble(dot(u, u) * dx)) * sqrt(1./domain_volume)
    nu_top = -1. * assemble(dot(grad(Tnew), n) * ds(top))
    Told.assign(Tnew)
```

Listing 1. Firedrake code required to reproduce 2-D Cartesian incompressible isoviscous benchmark cases from **?**.

Solutions are obtained by solving the Stokes equations for velocity and pressure, alongside the energy equation for temperature. The initial temperature distribution is prescribed as follows:

$$T(x,y) = (1-y) + A\cos(\pi x)\sin(\pi y), \tag{35}$$

where $A = 0.05$ is the amplitude of the initial perturbation.

We have set up the problem using a bilinear quadrilateral element pair (~~Q2-Q1~~Q2Q1) for velocity and pressure, with Q2 elements for temperature. Firedrake user code is written in Python, so the first step, illustrated on line 1 of Listing 1, is to import the Firedrake module. We next need a mesh: for simple domains such as the unit square, Firedrake provides built-in meshing functions. As such, line 4 defines the mesh, with 40 quadrilateral elements in x and y directions. We also need function spaces, which is achieved by associating the mesh with the relevant finite element on lines ~~7-9~~10-12: V, W and Q are symbolic variables representing function spaces. They also contain the ~~computational implementation of the function space~~function space's computational implementation, recording the association of degrees of freedom with the mesh and pointing to the finite element basis. The user does not usually need to pay any attention to this: the function space just behaves as a mathematical object (**?**). Function spaces can be combined in the natural way to create mixed function spaces, as we do on line ~~10~~13, combining the velocity and pressure function spaces to form a function space for the mixed Stokes problem, Z. ~~Note that although we use~~ Here we specify continuous Lagrange elements (CG) ~~in all examples presented herein, Firedrake offers a range of different options, including discontinuous elements (DG)~~of polynomial degree 2 and 1 for velocity and pressure respectively, on a quadrilateral mesh, which gives us the Q2Q1 element pair. Test functions, $\boldsymbol{v}$, w and q are subsequently defined (lines ~~13-14~~16-17) and we also specify functions to hold our solutions (lines ~~15-18~~18-21): z in the mixed function space, noting that a symbolic representation of the two parts – velocity and pressure – is obtained with `split` on line ~~16~~19, and T_{old} and T_{new} (line ~~17~~20), required for the ~~Crank-Nicholson~~ Crank-Nicolson scheme used for temporal discretisation in our energy equation (see Equations 19 and 20 in Section 4.2), where T_θ is defined on line ~~18.~~ 21.

We obtain symbolic expressions for coordinates in the physical mesh (line ~~21~~24) and subsequently use these to initialize the old temperature field, via Equation (35), on line ~~22. This is the first point at which~~ 25. This is where Firedrake transforms a symbolic operation into a numerical computation ~~. The~~ for the first time: the `interpolate` method generates C code ~~which~~ that evaluates this expression ~~at the nodes of~~ in the function space associated with T_{old}, and immediately executes it to populate the coefficient values of T_{old}. We initialize T_{new} with the values of T_{old}, on line ~~23~~26, via the `assign` function. Important constants in this problem (Rayleigh Number, Ra; viscosity, μ; thermal diffusivity, κ), ~~in addition to the constant timestep (Δ_t)~~ and unit vector (~~k~~$\hat{\boldsymbol{k}}$), are defined on lines ~~26-30.~~ 29-30. In addition we define a constant for timestep (Δ_t) with an initial value of 10^{-6}. `Constant` objects define spatial constants, with a value that can be overwritten in later timesteps, as we do in this example using an adaptive timestep. We note that viscosity could also be a `Function`, if we wanted spatial variation.

We are now in a position to define the variational problems expressed in Equations (25) and (19). Although in this test case the problems are linear, we maintain the more general nonlinear residual form $F_{\text{Stokes}}(\boldsymbol{v},\boldsymbol{u}) = 0$ and $F_{\text{energy}}(q,T) = 0$, to allow for straightforward extension to nonlinear problems below. The symbolic expressions for F_{Stokes} and F_{Energy} in the UFL are

given on lines 33-37: the resemblance to the mathematical formulation is immediately apparent. Integration over the domain is indicated by multiplication with `dx`.

~~Results from 2-D incompressible isoviscous square convection benchmark cases: (a) Nusselt number versus number of pressure and velocity degrees of freedom (DOF), at $Ra = 1 \times 10^4$ (Case 1a - **?**), for a series of uniform, structured meshes; (b) RMS velocity versus number of pressure and velocity DOF, at $Ra = 1 \times 10^4$; (c, d) as in panels a and b, but at $Ra = 1 \times 10^5$ (Case 1b - **?**); (e, f) at $Ra = 1 \times 10^6$ (Case 1c - **?**). Benchmark values are denoted by dashed red lines. In panels e and f, we also display results from simulations where temperature is represented through a Q1 discretisation (Q2Q1_Q1), for comparison with our standard Q2 temperature discretisations (Q2Q1_Q2).~~

Strong Dirichlet boundary conditions for velocity (bcvx, bcvy) and temperature (bctb, bctt) are specified on lines 40-41. A Dirichlet boundary condition is created by constructing a ~~Python~~ `DirichletBC` object, where ~~we~~ the user must provide the function space ~~the condition applies to, the~~ with the boundary condition value, and the part of the mesh at which it applies. The latter uses integer mesh markers which are commonly used by mesh generation software to tag entities of meshes. ~~The~~ Boundaries are automatically tagged by the built-in meshes supported by Firedrake~~(such as~~. For the `UnitSquareMesh` being used here~~) automatically tag the boundary. For this mesh~~, tag 1 corresponds to the plane $x = 0$; 2 to $x = 1$; 3 to $y = 0$; and 4 to $y = 1$ (these integer values are assigned to left, right, bottom and top on line 5). Note how ~~we are applying boundary conditions~~ boundary conditions are being applied to the velocity part of the mixed finite element space Z, indicated by `Z.sub(0)`. Within `Z.sub(0)` we can further subdivide into `Z.sub(0).sub(0)` and `Z.sub(0).sub(1)` to apply boundary conditions to the x and y components of the velocity field only. To apply conditions to the pressure space, we would use `Z.sub(1)`. This problem has a ~~nullspace of all functions of constant pressure , so we need to~~ constant pressure nullspace and we must ensure that our solver removes this space. To do so, we build a nullspace object on line 42, which will subsequently be passed to the solver, and PETSc will ~~take care of seeking~~ seek a solution in the space orthogonal to the provided nullspace.

We finally come to solving the variational problem, with problems and solver objects created on lines ~~59–62~~58–61. We pass in the residual functions F_{Stokes} and F_{Energy}, solution fields (z, T_{new}), boundary conditions and, for the Stokes system, the nullspace object. Solution of the two variational problems is undertaken by the PETSc library (**?**), guided by the solver parameters specified on lines ~~50–56~~ 50-55 (see **??**, for comprehensive documentation of all PETSc options). The first option on line 51, instructs the Jacobian to be assembled in PETSc's default `aij` sparse matrix type. Although the Stokes and energy problem in this example are linear, for consistency with latter cases, we use Firedrake's `NonlinearVariationalSolver` which makes use of PETSc's Scalable Nonlinear Equations Solvers (SNES) interface. However, since we do not actually need a nonlinear solver for this case, we choose the `ksponly` method on line 52 indicating that only a single linear solve needs to be performed. The linear solvers are configured through PETSc's Krylov Subspace (KSP) interface, where we can request a direct solver by choosing the `preonly` KSP method, in combination with `lu` as the 'preconditioner' (PC) type (lines ~~53–54~~53-54). The specific implementation of the LU-decomposition based direct solver is selected on line 55 ~~,~~ as the MUMPS library (**??**). As we shall see through subsequent examples, the solution process is fully programmable, enabling the creation of sophisticated solvers by ~~composing together~~ combining multiple layers of Krylov methods and preconditioners (**?**).

~~Final steady-state temperature field, in 2-D and 3-D, from Firedrake simulations, designed to match: (a) Case 1a from (?), with contours spanning temperatures of 0 to 1, at 0.05 intervals; (b) Case 1a from (?), with transparent isosurfaces plotted at $T = 0.3$, 0.5 and 0.7.~~

The time-loop is ~~initiated on line 64~~defined in lines 74-83, with the Stokes system solved on line ~~67~~ 79 and the energy equation on line ~~68.~~ 80. These `solve` calls once again convert symbolic mathematics into computation. The linear systems for both problems are based on the Jacobian matrix, and a right-hand side vector based on the residual, as indicated in Equations (22), (23) and (24) for the energy equation, and Equations (28), (29), (30) and (31) for the Stokes equation. Note, however, that the symbolic expression for the Jacobian is derived automatically in UFL. Firedrake's TSFC (**?**) subsequently converts the UFL into highly optimised assembly code~~. This code~~, which is then executed to create the matrix and vectors, ~~and~~ with the resulting system ~~is passed back~~ passed to PETSc for solution. ~~Finally, we note that output~~ Output is written on ~~line 66~~lines 77-78, to a *.pvd* file, initialised on line 45, for visualisation in software such as ParaView (e.g. **?**).

~~In < 70~~

After the first timestep the timestep size Δt is adapted (lines 75-76) to a value computed in the `compute_timestep` function (lines 68-71). This function computes a CFL-bound timestep by first computing the velocity transformed from physical coordinates into the local coordinates of the reference element. This transformation is performed by multiplying velocity by the inverse of the Jacobian of the physical coordinate transformation, and interpolating this into a predefined vector function `u_ref` (line 70). Since the dimensions of all quadrilaterals/hexahedra in local coordinates have unit length in each direction, the CFL condition now simplifies to $u_{\text{ref}} * \Delta t \leq 1$, which needs to be satisfied for all components of u_{ref}. The maximum allowable time step can thus be computed by extracting the reference velocity vectors at all nodal locations, obtained by taking the maximum of the `.dat.data` property of the interpolated function. The advantage of this method of computing the timestep over one based on the traditional CFL condition in the form of $u\Delta t/\Delta x \leq 1$, is that it generalizes to non-uniform and curved (iso-parametric) meshes.

In 83 lines of Python (56 excluding comments and blank lines), we are able to produce a model that can be executed and quantitatively compared with benchmark results from **?**. To do so, we have computed the RMS velocity ~~andsurface Nusselt number~~ (line 81, using the domain volume specified on line 7) and surface Nusselt number (line 82, using a unit normal vector defined on line 6), at a range of different mesh resolutions and Rayleigh numbers, with results presented in Figure 1. Results converge towards the benchmark solutions, with increasing resolution. The final steady-state temperature field, at $Ra = 1\times10^6$, is illustrated in Figure 2(a).

To further highlight the flexibility of Firedrake, we have also simulated some of these cases using a $\text{Q2P}_{1\text{DG}}$ discretisation for the Stokes system and a Q1 discretisation for the temperature field. The modifications necessary are minimal: for the former, on line 9, the ~~polynomial order is~~ finite element family is specified as 'DPC', which instructs Firedrake to use a discontinuous, piecewise linear discretisation for pressure. Note that this choice is distinct from a discontinuous, piecewise bilinear pressure space, which, in combination with Q2 velocities, is not LBB stable, whereas the $\text{Q2P}_{1\text{DG}}$ pair is (**?**). For temperature, the degree specified on line 10 is changed from 2 to 1. Results ~~, at~~ using a discontinuous linear pressure, at $Ra = 1\times10^5$, are presented in Figure 1(c,d), showing a similar trend to those of the Q2Q1 element pair, albeit with RMS velocities converging

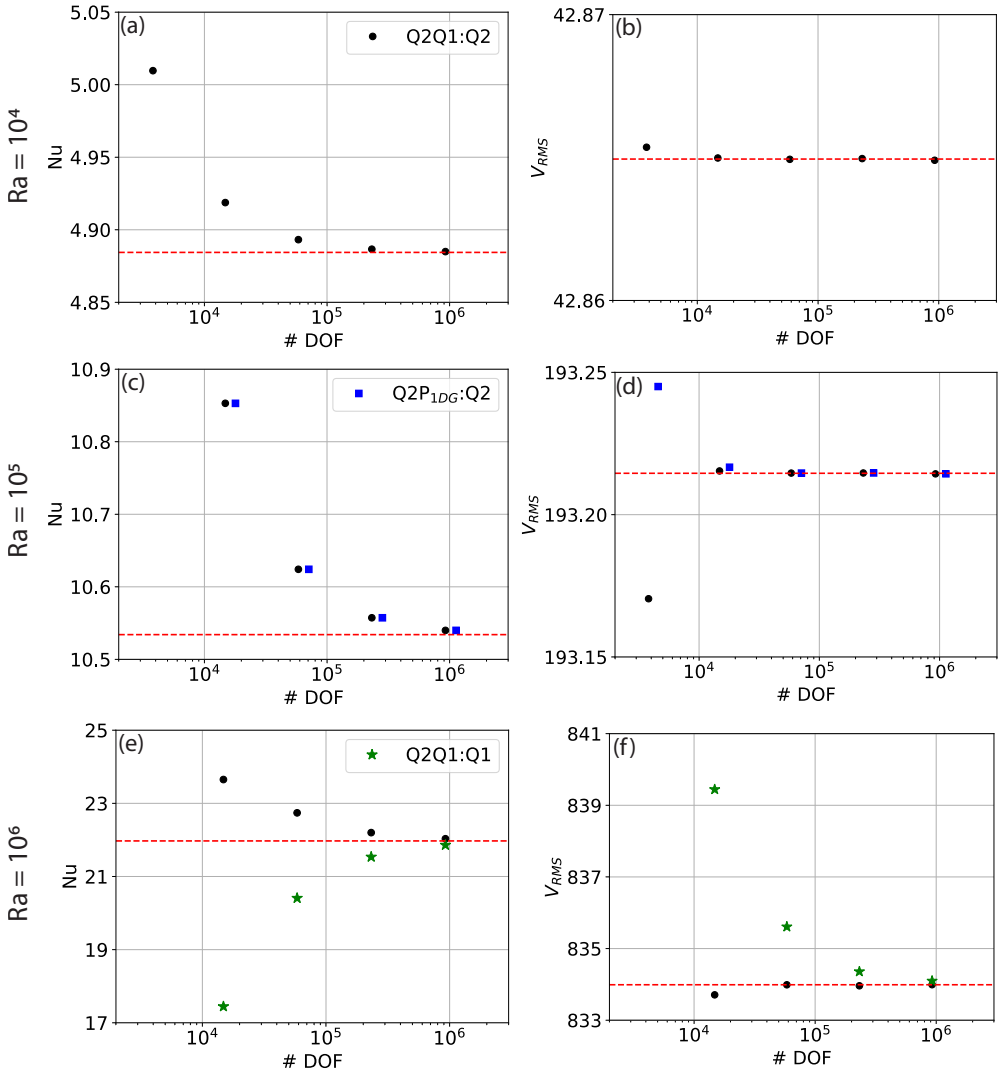

Figure 1. Results from 2-D incompressible isoviscous square convection benchmark cases: (a) Nusselt number versus number of pressure and velocity degrees of freedom (DOF), at $Ra = 1 \times 10^4$ (Case 1a - **?**), for a series of uniform, structured meshes; (b) RMS velocity versus number of pressure and velocity DOF, at $Ra = 1 \times 10^4$; (c, d) as in panels a and b, but at $Ra = 1 \times 10^5$ (Case 1b - **?**); (e, f) at $Ra = 1 \times 10^6$ (Case 1c - **?**). Benchmark values are denoted by dashed red lines. In panels c and d, we also display results from simulations where the Stokes system uses the $Q2P_{1DG}$ finite element pair ($Q2P_{1DG} : Q2$), and in panels e and f where temperature is represented using a Q1 discretisation (Q2Q1:Q1), for comparison with our standard Q2Q1:Q2 discretisations.

towards benchmark values from above rather than below. Results using a Q1 discretisation for temperature, at $Ra = 1 \times 10^6$, are presented in Figure 1(e,f), ~~again~~ converging towards benchmark values with increasing resolution. We find that, as expected, a Q2 temperature discretisation leads to more accurate results, although results converge towards the benchmark solutions from

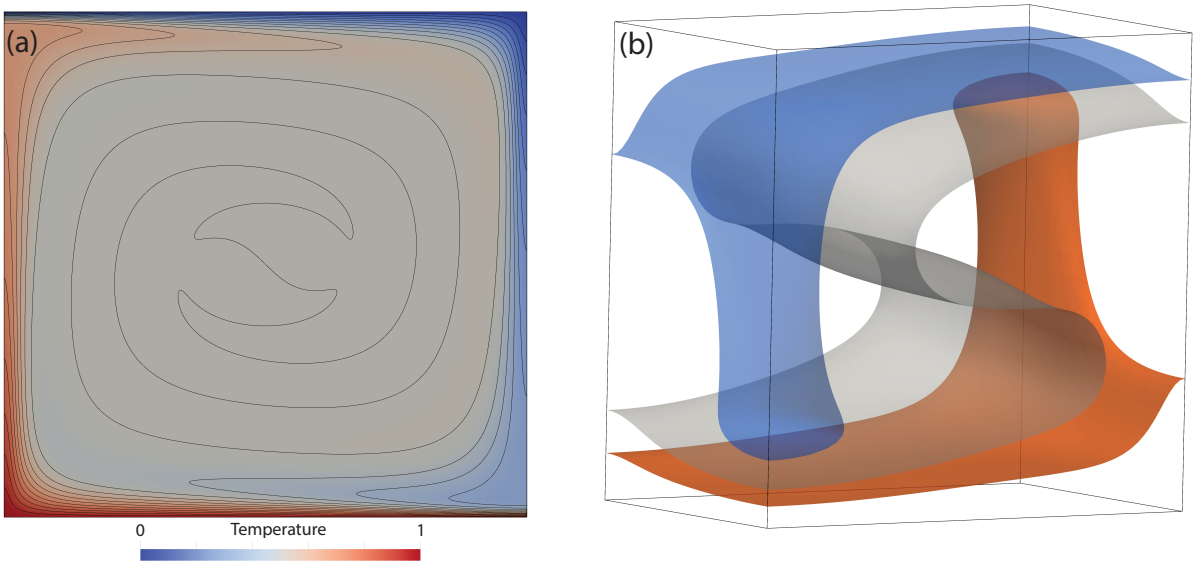

Figure 2. Final steady-state temperature field, in 2-D and 3-D, from Firedrake simulations, designed to match: (a) Case 1a from (**?**), with contours spanning temperatures of 0 to 1, at 0.05 intervals; (b) Case 1a from (**?**), with transparent isosurfaces plotted at $T = 0.3$, 0.5 and 0.7.

different directions. For the remainder of the examples considered herein, we use a Q2Q1 discretisation for the Stokes system and a Q2 discretisation for temperature.

5.2 Extension: more realistic physics

We next highlight the ease at which simulations can be updated to incorporate more realistic physical approximations. We first account for compressibility, under the Anelastic Liquid Approximation (e.g. **?**), simulating a well-established benchmark case from ~~?~~ **?** (Section 5.2.1). We subsequently focus on a case with a more Earth-like approximation of the rheology (Section 5.2.2), simulating another well-established benchmark case from **?**. All cases are set up in an enclosed 2-D Cartesian box with free-slip boundary conditions, with the required changes discussed relative to the base case presented in Section 5.1.

5.2.1 Compressibility

The governing equations applicable for compressible mantle convection, under the Anelastic Liquid Approximation (ALA), are presented in Appendix **??** (based on, for example, **?**). Their weak forms are derived by multiplying these equations with appropriate test functions and integrating over the domain, as we did with their incompressible counterparts in Section 4. They differ appreciably from the incompressible approximations that have been utilised thus far, with important updates to all three governing equations. Despite this, the changes required to incorporate these equations, within UFL and Firedrake, are minimal.

~~Results from Firedrake simulations configured to reproduce 2-D compressible benchmark case from ? at $Ra = 10^5$ and $Di = 0.5$: (a) final steady-state (full) temperature field, with contours spanning temperatures of 0 to 1, at 0.05 intervals; (b) Nusselt number versus number of pressure and velocity DOF, for a series of uniform, structured meshes; (c) RMS velocity versus number of pressure and velocity DOF. The range of solutions provided by different codes in the ? benchmark study are bounded by dashed red lines.~~

~~Although ?~~ Although **?** examined a number of cases, we focus on one illustrative example here, at $Ra = 10^5$ and a dissipation number $Di = 0.5$. This allows us to demonstrate the ease at which these cases can be configured within Firedrake. The required changes, relative to the base case, are displayed in Listing 2. They can be summarised as follows:

1. Definition and initialisation of additional constants and the 1-D reference state, derived here via an Adams-Williamson equation of state (lines 1-12). In this benchmark example, several of the key constants and parameters required for compressible convection are assigned values of 1 and could be removed. However, to ensure consistency between the governing equations presented in Appendix **??** and the UFL, we chose not to omit these constants in Listing 2.

2. The UFL for the momentum, mass conservation and energy equations is updated, emphasising once again the resemblance to the mathematical formulation (lines 16-20). The key changes are as follows: (i) the stress tensor is updated to account for a non-zero velocity divergence (line 17), where `Identity` represents a unit matrix of a given size (2 in this case) and `div` represents the symbolic divergence of a field; (ii) the Stokes equations are further modified to account for dynamic pressure's influence on buoyancy (final term on line 18); (iii) the mass conservation equation includes the depth-dependent reference density, $\bar{\rho}$ (line 19); and (iv) the energy equation is updated to incorporate adiabatic heating and viscous dissipation terms (final 2 terms on line 20).

```
1  # Additional constants and definition of compressible reference state:
2  Ra = Constant(1e5) # Rayleigh number
3  Di = Constant(0.5) # Dissipation number
4  T0 = Constant(0.091) # Non-dimensional surface temperature
5  tcond = Constant(1.0) # Thermal conductivity
6  rho_0, alpha, cpr, cvr, gruneisen = 1.0, 1.0, 1.0, 1.0, 1.0
7  rhobar = Function(Q, name="CompRefDensity").interpolate(rho_0 * exp(((1.0 - X[1]) * Di) / alpha))
8  Tbar = Function(Q, name="CompRefTemperature").interpolate(T0 * exp((1.0 - X[1]) * Di) - T0)
9  alphabar = Function(Q, name="IsobaricThermalExpansivity").assign(1.0)
10 cpbar = Function(Q, name="IsobaricSpecificHeatCapacity").assign(1.0)
11 chibar = Function(Q, name="IsothermalBulkModulus").assign(1.0)
12 FullT = Function(Q, name="FullTemperature").assign(Tnew+Tbar)
13
14 ----------------------------------------------------------------------------------------
15 # Equations in UFL:
16 I = Identity(2)
17 stress = 2 * mu * sym(grad(u)) - 2./3.*I*mu*div(u)
18 F_stokes = inner(grad(v), stress) * dx - div(v) * p * dx - (dot(v, k) * (Ra * Ttheta * rhobar * alphabar - (Di/
       gruneisen) * (cpr/cvr)*rhobar*chibar*p) * dx)
19 F_stokes += -w * div(rhobar*u) * dx  # Mass conservation
20 F_energy = q * rhobar * cpbar * ((Tnew - Told) / delta_t) * dx + q * rhobar * cpbar * dot(u, grad(Ttheta)) * dx +
        dot(grad(q), tcond * grad(Tbar + Ttheta)) * dx + q * (alphabar * rhobar * Di * u[1] * Ttheta) * dx  - q * (
        (Di/Ra) * inner(stress, grad(u)) ) * dx
21
22 ----------------------------------------------------------------------------------------
23 # Temperature boundary conditions:
24 bctb, bctt = DirichletBC(Q, 1.0 - (T0*exp(Di) - T0), bottom), DirichletBC(Q, 0.0, top)
25
26 ----------------------------------------------------------------------------------------
27 # Pressure nullspace:
28 stokes_solver = NonlinearVariationalSolver(stokes_problem, solver_parameters=solver_parameters,
       transpose_nullspace=p_nullspace)
```

Listing 2. Difference in Firedrake code required to reproduce compressible ALA cases from **?** relative to our base case.

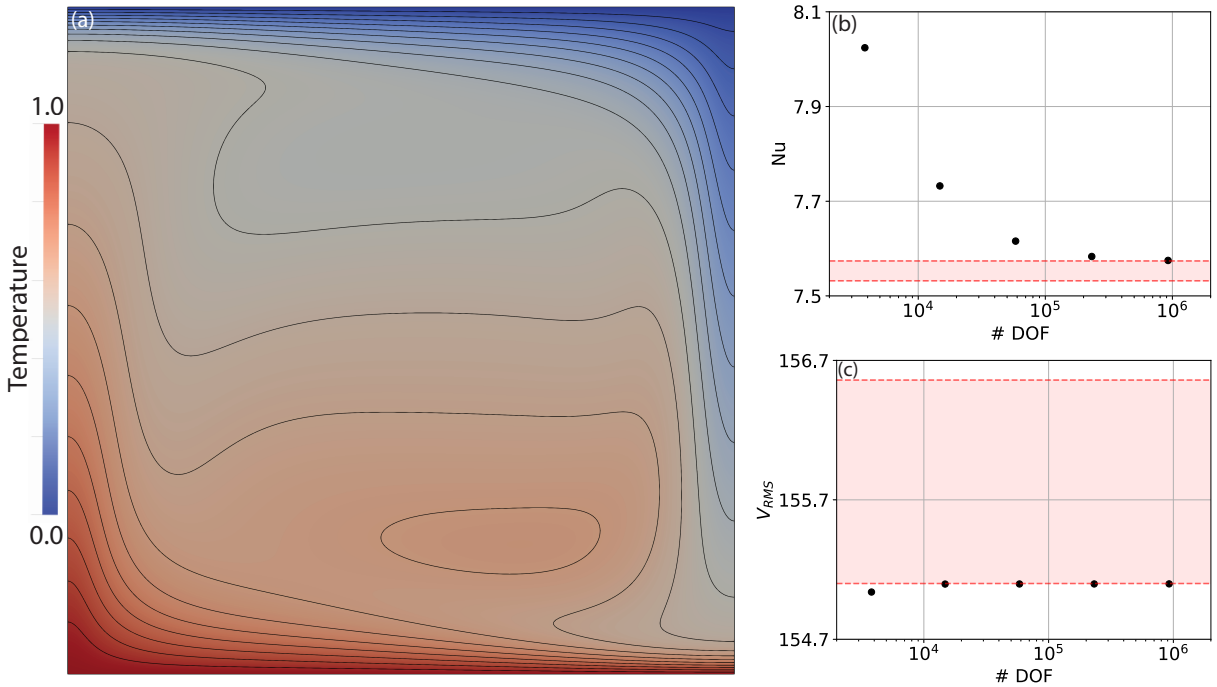

Figure 3. Results from Firedrake simulations configured to reproduce 2-D compressible benchmark case from **?** at $Ra = 10^5$ and $Di = 0.5$: (a) final steady-state (full) temperature field, with contours spanning temperatures of 0 to 1, at 0.05 intervals; (b) Nusselt number versus number of pressure and velocity DOF, for a series of uniform, structured meshes; (c) RMS velocity versus number of pressure and velocity DOF. The range of solutions provided by different codes in the **?** benchmark study are bounded by dashed red lines.

3. Temperature boundary conditions are updated, noting that we are solving for deviatoric temperature rather than the full temperature, which also includes the reference state.

4. In our Stokes solver, we only specify the `transpose_nullspace` option (as opposed to both `nullspace` and `transpose_nullspace` options for our base case): the incorporation of dynamic pressure's impact on buoyancy implies that the (right-hand side) pressure nullspace is no longer the same as the (left-hand side) transpose nullspace. The transpose nullspace remains the same space of constant pressure solutions, and is used to project out these modes from the initial residual vector to ensure that the linear system is well-posed. The right-hand side nullspace now consists of different modes, which can be found through integration. However, this nullspace is only required for iterative linear solvers in which the modes are projected out from the solution vector at each iteration to prevent its unbounded growth.

We note that in setting up the Stokes solver as we have, we incorporate the pressure effect on buoyancy implicitly, as advocated by **?**. As this term depends on the pressure that we are solving for, an extra term is required in addition to the pressure gradient matrix G in the Jacobian matrix in Equation (28). The inclusion of $\bar{\rho}$ in the continuity constraint also means that this term is no longer simply represented by the transpose of G. Such changes are automatically incorporated by Firedrake,

```
# Stokes solver dictionary:
stokes_solver_parameters = {
    "mat_type": "aij",
    "snes_type": "newtonls",
    "snes_linesearch_type": "l2",
    "snes_max_it": 100,
    "snes_atol": 1e-10,
    "ksp_type": "preonly",
    "pc_type": "lu",
    "pc_factor_mat_solver_type": "mumps",
}

# Energy solver dictionary:
energy_solver_parameters = {
    "mat_type": "aij",
    "snes_type": "ksponly",
    "ksp_type": "preonly",
    "pc_type": "lu",
    "pc_factor_mat_solver_type": "mumps",
}

-----------------------------------------------------------------------------------------------
# Viscosity calculation and Rayleigh number:
Ra = Constant(100.) # Rayleigh number
gamma_T, gamma_Z = Constant(ln(10**5)), Constant(ln(10))
mu_star, sigma_y = Constant(0.001), Constant(1.0)
epsilon = sym(grad(u))  # Strain-rate
epsii = sqrt(inner(epsilon,epsilon) + 1e-20) # 2nd invariant (with tolerance to ensure stability)
mu_lin = exp(-gamma_T*Tnew + gamma_Z*(1 - X[1]))
mu_plast = mu_star + (sigma_y / epsii)
mu = (2. * mu_lin * mu_plast) / (mu_lin + mu_plast)

-----------------------------------------------------------------------------------------------
# Updated solver:
stokes_solver = NonlinearVariationalSolver(stokes_problem, solver_parameters=stokes_solver_parameters, nullspace=
    p_nullspace, transpose_nullspace=p_nullspace)
energy_solver = NonlinearVariationalSolver(energy_problem, solver_parameters=energy_solver_parameters)
```

Listing 3. Difference in Firedrake code required to reproduce viscoplastic rheology cases from **?** relative to our base case.

highlighting a major benefit of the automatic assembly approach that is utilised. To ensure the validity of our approach, we have computed the RMS velocity and Nusselt number at a range of different mesh resolutions, for direct comparison with ~~?~~?, with results presented in Figure 3, alongside the final steady-state (full) temperature field. As expected, results converge towards the benchmark solutions, with increasing resolution, demonstrating the applicability and accuracy of Firedrake for compressible simulations of this nature.

5.2.2 Viscoplastic Rheology

~~Results from 2-D benchmark case from **?**, with a viscoplastic rheology, at $Ra_0 = 10^2$: (a) Nusselt number versus number of pressure and velocity DOF, for a series of uniform, structured meshes; (b) final steady-state temperature field, with contours spanning temperatures of 0 to 1, at 0.05 intervals; (c) RMS velocity versus number of pressure and velocity DOF; (d) final steady-state viscosity field (note logarithmic scale). In panels a and c, the range of solutions provided by different codes in the **?** benchmark study are bounded by dashed red lines.~~

To illustrate the changes necessary to incorporate a viscoplastic rheology, which is more representative of deformation within Earth's mantle and lithosphere, we examine a case from **?**, a benchmark study intended to form a straightforward extension to **?**. Indeed, aside from the viscosity and reference Rayleigh Number ($Ra_0 = 10^2$), all other aspects of this case are identical to the case presented in Section 5.1. The viscosity field, μ, is calculated as the harmonic mean between a linear component, μ_{lin}

and a nonlinear, plastic component, μ_{plast}, which is dependent on the strain-rate, as follows:

$$\mu(T, z, \dot{\epsilon}) = 2\left(\frac{1}{\mu_{\text{lin}(T,z)}} + \frac{1}{\mu_{\text{plast}(\dot{\epsilon})}}\right)^{-1}. \tag{36}$$

The linear part is given by an Arrhenius law (the so-called Frank-Kamenetskii approximation):

$$\mu_{\text{lin}(T,z)} = \exp(-\gamma_T T + \gamma_z z), \tag{37}$$

where $\gamma_T = \ln(\Delta\mu_T)$ and $\gamma_z = \ln(\Delta\mu_z)$ are parameters controlling the total viscosity contrast due to temperature and depth, respectively. The nonlinear component is given by:

$$\mu_{\text{plast}}(\dot{\epsilon}) = \mu^\star + \frac{\sigma_y}{\sqrt{\dot{\epsilon} : \dot{\epsilon}}} \tag{38}$$

where $\mu^\star$ is a constant representing the effective viscosity at high stresses and σ_y is the yield stress. The denominator of the second term in Equation (38) represents the second invariant of the strain-rate tensor. The viscoplastic flow law (Eq. 36) leads to linear viscous deformation at low stresses and plastic deformation at stresses that exceed σ_y, with the decrease in viscosity limited by the choice of $\mu^\star$.

Although **?** examined a number of cases, we focus on one here (Case 4: $Ra_0 = 10^2$, $\Delta\mu_T = 10^5$, $\Delta\mu_y = 10$ and $\mu^\star = 10^{-3}$), which allows us to demonstrate how a temperature-, depth- and strain-rate dependent viscosity is incorporated within Firedrake. The changes required to simulate this case, relative to our base case, are displayed in Listing 3. These are:

1. Linear solver options are no longer applicable, given the dependence of ~~of~~ viscosity on the flow field, through the strain-rate. Accordingly, the solver dictionary is updated to account for the nonlinear nature of our Stokes system (lines 2-11). For the first time, we fully-exploit the SNES, using a setup based on Newton's method (`"snes_type": "newtonls"`) with a secant line search over the L2-norm of the function (`"snes_linesearch_type": "l2"`). As we target a steady-state solution, an absolute tolerance is specified for our nonlinear solver (`"snes_atol": 1e-10`).

2. Solver options differ between the (nonlinear) Stokes and (linear) energy systems. As such, a separate solver dictionary is specified for solution of the energy equation (lines 13-20). Consistent with our base case, we use a direct solver for solution of the energy equation, based on the ~~Mumps~~ MUMPS library.

3. Viscosity is calculated as a function of temperature, depth (μ_{lin} - line 29) and strain-rate (μ_{plast} - line 30), using constants specified on lines 25-26. Linear and nonlinear components are subsequently combined via a harmonic mean (line 31).

4. Updated solver dictionaries are incorporated into their respective solvers on lines 35 and 36, noting that for this case both the nullspace and transpose_nullspace options are provided for the Stokes system, consistent with the base case.

We note that even though the UFL for the Stokes and energy systems remains identical to our base case, assembly of additional terms in the Jacobian, associated with the nonlinearity in this system, is once again handled automatically by Firedrake. To compare our results to those of **?** we have computed the RMS velocity and Nusselt number at a range of different mesh

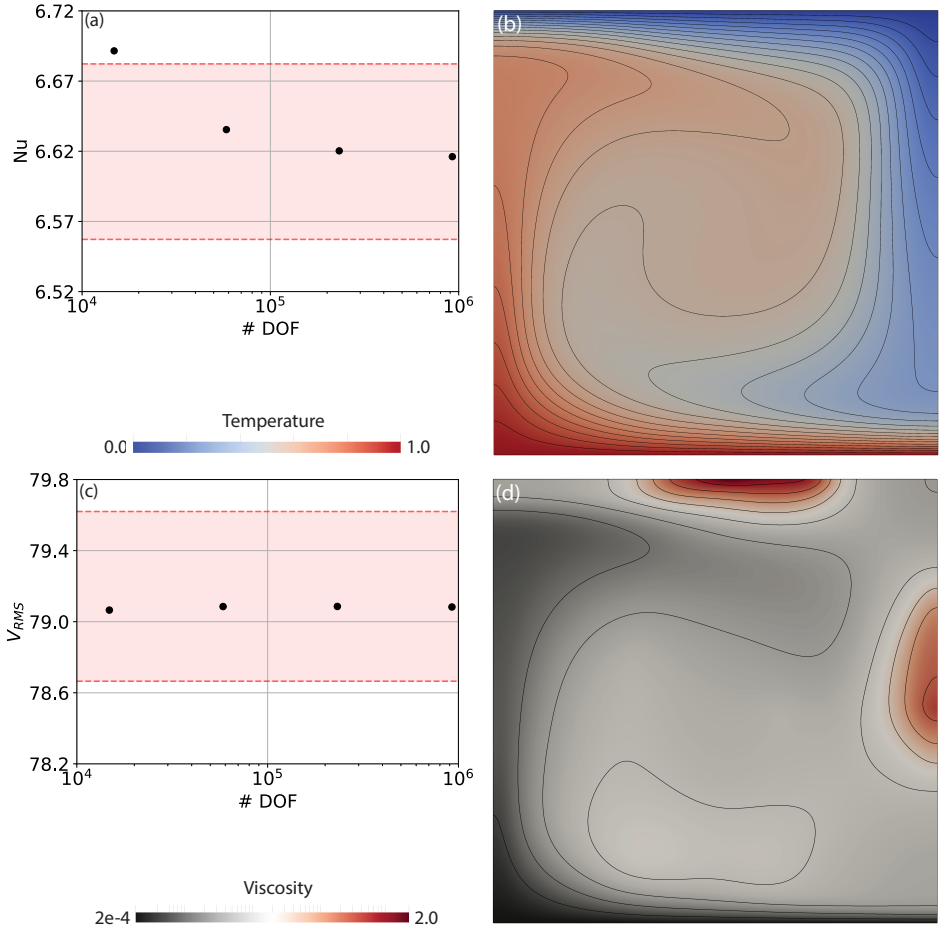

Figure 4. Results from 2-D benchmark case from **?**, with a viscoplastic rheology, at $Ra_0 = 10^2$: (a) Nusselt number versus number of pressure and velocity DOF, for a series of uniform, structured meshes; (b) final steady-state temperature field, with contours spanning temperatures of 0 to 1, at 0.05 intervals; (c) RMS velocity versus number of pressure and velocity DOF; (d) final steady-state viscosity field (note logarithmic scale). In panels a and c, the range of solutions provided by different codes in the **?** benchmark study are bounded by dashed red lines.

resolutions. These are presented in Figure 4 and, once again, results converge towards the benchmark solutions, with increasing resolution. Final steady-state temperature and viscosity fields are also illustrated to allow for straightforward comparison with those presented by **?**, illustrating that viscosity varies by roughly four orders of magnitude across the computational domain.

Taken together, our compressible and viscoplastic rheology results demonstrate the accuracy and applicability of Firedrake for problems incorporating a range of different approximations to the underlying physics. They have allowed us to illustrate Firedrake's flexibility: by leveraging UFL and PETSc, the framework is easily extensible, allowing for straightforward application to scenarios involving different physical approximations, even if they require distinct solution strategies.

5.3 Extension: Dimensions and Geometry

In this section we highlight the ease at which simulations can be examined in different dimensions and geometries, by modifying our basic 2-D case. We primarily simulate benchmark cases that are well-known within the geodynamical community, initially matching the steady-state, isoviscous simulation of **?** in a 3-D Cartesian domain. There is currently no published community benchmark for simulations in the 2-D cylindrical shell domain. As such, we next compare results for an isoviscous, steady-state case, in a 2-D cylindrical shell domain, with those of the Fluidity ~~computational modelling framework (?), which~~ and ASPECT computational modelling frameworks, noting that Fluidity has been carefully validated against the extensive set of analytical solutions introduced by **?**, in both cylindrical and spherical shell geometries. Finally, we analyze an isoviscous 3-D spherical shell benchmark case from **?**. Once again, the changes required to run these cases are discussed relative to our base case (Section 5.1), unless noted otherwise.

5.3.1 3-D Cartesian Domain

We first examine and validate our setup in a 3-D Cartesian domain, for a steady-state, isoviscous case – specifically Case 1a from **?**. The domain is a box of dimensions $1.0079 \times 0.6283 \times 1$. The initial temperature distribution, chosen to produce a single ascending and descending flow, at $x = y = 0$ and $(x = 1.0079, y = 0.6283)$, respectively, is prescribed as:

$$T(x,y,z) = \left[\frac{\mathrm{erf}(4(1-z)) + \mathrm{erf}(-4z) + 1}{2}\right] + A[\cos(\pi x/1.0079) + \cos(\pi y/0.6283)]\sin(\pi z), \tag{39}$$

where $A = 0.2$ is the amplitude of the initial perturbation. We note that this initial condition differs to that specified in **?**, through the addition of boundary layers at the bottom and top of the domain (through the erf terms), although it more consistently drives solutions towards the final published steady-state results. Boundary conditions for temperature are T = 0 at the surface (z = 1) and T = 1 at the base (z = 0), with insulating (homogeneous Neumann) sidewalls. No-slip velocity boundary conditions are specified at the top surface and base of the domain, with free-slip boundary conditions on all sidewalls. The Rayleigh number $Ra = 3 \times 10^4$.

In comparison to Listing 1, the changes required to simulate this case, using ~~trilinear (Q2-Q1)~~ Q2Q1 elements for velocity and pressure, are minimal. The key differences, summarised in Listing 4, are:

1. The creation of the underlying mesh (lines 1-5), which we generate by extruding a 2-D quadrilateral mesh in the z-direction to a layered 3-D hexahedral mesh. Our final mesh has $20 \times 12 \times 20$ elements, in x-, y- and z-directions, respectively (noting that the default value for layer height is 1 / nz). For extruded meshes, top and bottom boundaries are tagged by `top` and `bottom`, respectively, whilst boundary markers from the base mesh can be used to set boundary conditions on the relevant side of the extruded mesh. We note that Firedrake exploits the regularity of extruded meshes to enhance performance.

2. Specification of the initial condition for temperature, following Equation (39), updated values for Ra, and definition of the 3-D unit vector (lines 9-11).

```
1 # Mesh Generation:
2 a, b, c, nx, ny, nz = 1.0079, 0.6283, 1.0, 20, int(0.6283/1.0 * 20), 20
3 mesh2d = RectangleMesh(nx, ny, a, b, quadrilateral=True) # Rectangular 2D mesh
4 mesh = ExtrudedMesh(mesh2d, nz)
5 bottom, top, left, right, front, back = "bottom", "top", 1, 2, 3, 4
6
7 ----------------------------------------------------------------------------------------------
8 # Initial condition and constants:
9 Told.interpolate(0.5*(erf((1-X[2])*4)+erf(-X[2]*4)+1) + 0.2*(cos(pi*X[0]/a)+cos(pi*X[1]/b))*sin(pi*X[2]))
10 Ra = Constant(3e4) # Rayleigh number
11 k = Constant((0, 0, 1)) # Unit vector (in direction opposite to gravity).
12
13 ----------------------------------------------------------------------------------------------
14 # Stokes Equation Solver Parameters:
15 stokes_solver_parameters = {
16     "mat_type": "matfree",
17     "snes_type": "ksponly",
18     "ksp_type": "preonly",
19     "pc_type": "fieldsplit",
20     "pc_fieldsplit_type": "schur",
21     "pc_fieldsplit_schur_type": "full",
22     "fieldsplit_0": {
23         "ksp_type": "cg",
24         "ksp_rtol": 1e-7,
25         "pc_type": "python",
26         "pc_python_type": "firedrake.AssembledPC",
27         "assembled_pc_type": "gamg",
28         "assembled_pc_gamg_threshold": 0.01,
29         "assembled_pc_gamg_square_graph": 100,
30     },
31     "fieldsplit_1": {
32         "ksp_type": "fgmres",
33         "ksp_rtol": 1e-6,
34         "pc_type": "python",
35         "pc_python_type": "firedrake.MassInvPC",
36         "Mp_ksp_rtol": 1e-5,
37         "Mp_ksp_type": "cg",
38         "Mp_pc_type": "sor",
39     } }
40
41 # Energy Equation Solver Parameters:
42 energy_solver_parameters = {
43     "mat_type": "aij",
44     "snes_type": "ksponly",
45     "ksp_type": "gmres",
46     "ksp_rtol": 1e-7,
47     "pc_type": "sor", }
48
49 ----------------------------------------------------------------------------------------------
50 # Set up boundary conditions:
51 bcvfb = DirichletBC(Z.sub(0).sub(1), 0, (front, back))
52 bcvlr = DirichletBC(Z.sub(0).sub(0), 0, (left, right))
53 bcvbt = DirichletBC(Z.sub(0), 0, (bot,top))
54 bctb, bctt = DirichletBC(Q, 1.0, bot), DirichletBC(Q, 0.0, top)
55
56 ----------------------------------------------------------------------------------------------
57 # Generating near_nullspaces for GAMG:
58 x_rotV = Function(V).interpolate(as_vector((0, X[2], -X[1])))
59 y_rotV = Function(V).interpolate(as_vector((-X[2], 0, X[0])))
60 z_rotV = Function(V).interpolate(as_vector((-X[1], X[0], 0)))
61 nns_x = Function(V).interpolate(Constant([1., 0., 0.]))
62 nns_y = Function(V).interpolate(Constant([0., 1., 0.]))
63 nns_z = Function(V).interpolate(Constant([0., 0., 1.]))
64 V_near_nullspace = VectorSpaceBasis([nns_x, nns_y, nns_z, x_rotV, y_rotV, z_rotV])
65 V_near_nullspace.orthonormalize()
66 Z_near_nullspace = MixedVectorSpaceBasis(Z, [V_near_nullspace, Z.sub(1)])
67
68 ----------------------------------------------------------------------------------------------
69 # Updated solve setup:
70 stokes_problem = NonlinearVariationalProblem(F_stokes, z, bcs=[bcvbt, bcvfb, bcvlr])
71 stokes_solver = NonlinearVariationalSolver(stokes_problem, solver_parameters=stokes_solver_parameters, appctx={"
       mu": mu}, nullspace=p_nullspace, transpose_nullspace=p_nullspace, near_nullspace=Z_near_nullspace)
72 energy_problem = NonlinearVariationalProblem(F_energy, Tnew, bcs=[bctb, bctt])
73 energy_solver = NonlinearVariationalSolver(energy_problem, solver_parameters=energy_solver_parameters)
74
75 ----------------------------------------------------------------------------------------------
76 # Updated diagnostics:
77 nusselt_number_top = -1. * assemble(dot(grad(Tnew), n) * ds_t) * (1./assemble(Tnew * ds_b))
```

Listing 4. Changes required to reproduce a 3-D Cartesian case from **?** relative to Listing 1.

3. The inclusion of Python dictionaries that define iterative solver parameters for the Stokes and energy systems (lines 15-47). Although direct solves provide robust performance in the 2-D cases examined above, in 3-D the computational (CPU and memory) requirements quickly become intractable. PETSc's `fieldsplit` `pc_type` provides a class of preconditioners for mixed problems that allows one to apply different preconditioners to different blocks of the system. This opens up

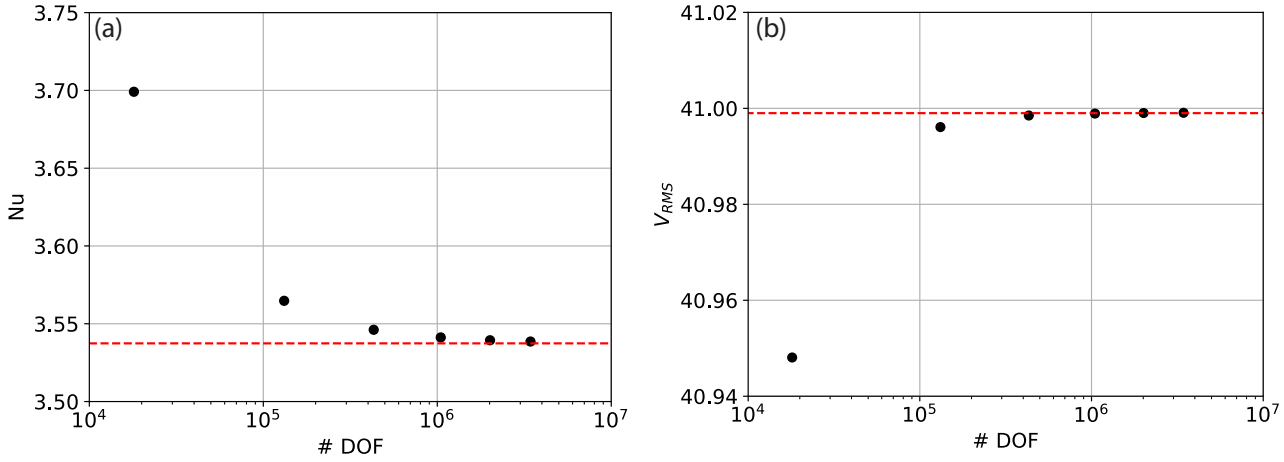

Figure 5. Results from 3-D isoviscous simulations in Firedrake, configured to reproduce benchmark results from Case 1a of **?**: (a) Nusselt number vs. number of pressure and velocity degrees of freedom (DOF), at $Ra = 3 \times 10^4$, for a series of uniform, structured meshes; (b) RMS velocity vs. number of pressure and velocity DOF. Benchmark values are denoted by dashed red lines.

a large array of potential solver strategies for the Stokes saddle point system (e.g. many of the methods described in **?**). Here we configure the Schur complement approach as described in Section 4.3. We note that this `fieldsplit` functionality can also be used to provide a stronger coupling between the Stokes system and energy equation in strongly nonlinear problems, where the Stokes and energy system are solved together in single Newton solve that is decomposed through a series of preconditioner stages.

The `fieldsplit_0` entries configure solver options for the first of these blocks, the K matrix. The linear systems associated with this matrix are solved using a combination of the Conjugate Gradient method (`cg`, line 23) and an algebraic multigrid preconditioner (`gamg`, line 27). We also specify two options (`gamg_threshold` and `gamg_square_graph`) that control the aggregation method (coarsening strategy) in the GAMG preconditioner, which balance the multigrid effectiveness (convergence rate) with coarse grid complexity (cost per iteration) (**?**).

The `fieldsplit_1` entries contain solver options for the Schur complement solve itself. As explained in Section 4.3 we do not have explicit access to the Schur complement matrix, $\mathrm{G}^T\mathrm{K}^{-1}\mathrm{G}$, but can compute its action on any vector, at the cost of a `fieldsplit_0` solve with the K matrix, which is sufficient to solve the system using a Krylov method. However, for preconditioning, we do need access to the values of the matrix or its approximation. For this purpose we approximate the Schur complement matrix with a mass matrix scaled by viscosity, which is implemented in `MassInvPC` (line 35) with the viscosity provided through the optional `appctx` argument on line 71. This is a simple example of Firedrake's powerful programmable preconditioner interface which, in turn, connects with the Python preconditioner interface of PETSc (line 34). In more complex cases the user can specify their own linear operator in UFL that approximates the true linear operator but is easier to invert. The `MassInvPC` preconditioner step itself is performed through a linear solve with the approximate matrix with options prefixed with `Mp_` to specify a Conjugate Gradient solver with symmetric SOR

(SSOR) preconditioning (lines 36-38). Note that PETSc's `sor` preconditioner type, specified on line 38, defaults to the symmetric SOR variant. Since this preconditioner step now involves an iterative solve, the Krylov method used for the Schur complement needs to be of flexible type, and we specify ~~flexible GMRES (~~`fgmres`~~)~~ on line 32.

Specification of the matrix type `matfree` (line 16) for the combined system ensures that we do not explicitly assemble its associated sparse matrix, instead computing the matrix-vector multiplications required by the Krylov iterations as they arise. For example the action of the submatrix G on a subvector $\underline{p}$ can be evaluated as (cf. Equations 28, 30):

$$G\underline{p} = \sum_k G_{ik} p_k = -\int_\Omega (\nabla \cdot \boldsymbol{\psi}_i)\, p \, \mathrm{d}\boldsymbol{x} \tag{40}$$

which is assembled by Firedrake directly from the symbolic expression into a discrete vector. Again, for preconditioning in the K-matrix solve we need access to matrix values, which is achieved using `AssembledPC`. This explicitly assembles the K-matrix by extracting relevant terms from the `F_Stokes` form.

Finally, the energy solve is performed through a combination of the GMRES (`gmres`) Krylov method and SSOR preconditioning (lines 42-47). For all iterative solves we specify a convergence criterion based on the relative reduction of the preconditioned residual (`ksp_rtol`: lines 24, 33, 36 and 46).

4. Velocity boundary conditions, which must be specified along all 6 faces, are modified on lines 51-53, with temperature boundary conditions specified on line 54.

5. Generating near-nullspace information for the GAMG preconditioner (lines 58-66), consisting of three rotational (x_rotV, y_rotV, z_rotV) and three translational (nns_x, nns_y, nns_z) modes, as outlined in Section 4.3. These are combined in the mixed function space on line 66.

6. Updating of the Stokes problem (line 70) to account for additional boundary conditions, and the Stokes solver (line 71) to include the near nullspace options defined above, in addition to the optional `appctx` keyword argument that passes the viscosity through to our `MassInvPC` Schur complement preconditioner. Energy solver options are also updated relative to our base case (lines 72-73), using the dictionary created on lines 42-47.

Our model results can be validated against those of **?**. As with our previous examples, we compute the Nusselt number and RMS velocity at a range of different mesh resolutions, with results presented in Figure 5. We find that results converge towards the benchmark solutions, with increasing resolution, as expected. The final steady state temperature field is illustrated in Figure 2(b).

5.3.2 2-D Cylindrical Shell Domain

We next examine simulations in a 2-D cylindrical ~~domain. The domain is~~ shell domain, defined by the radii of the inner (r_{min}) and outer (r_{max}) boundaries. These are chosen such that the non–dimensional depth of the mantle, $z = r_{\text{max}} - r_{\text{min}} = 1$, and the ratio of the inner and outer radii, $f = r_{\text{min}}/r_{\text{max}} = 0.55$, thus approximating the ratio between the radii of Earth's surface and

core-mantle-boundary (CMB). Specifically, we set $r_{\text{min}} = 1.22$ and $r_{\text{max}} = 2.22$. The initial temperature distribution, chosen to produce 4 equidistant plumes, is prescribed as:

$$T(x,y) = (r_{\text{max}} - r) + A\cos(4\ \text{atan2}(y,x))\sin(r - r_{\text{min}})\pi) \tag{41}$$

where $A = 0.02$ is the amplitude of the initial perturbation. Boundary conditions for temperature are $T = 0$ at the surface (r_{max}) and $T = 1$ at the base (r_{min}). Free-slip velocity boundary conditions are specified on both boundaries, which we incorporate weakly through the Nitsche approximation (see Section 4.1). The Rayleigh number $Ra = 1 \times 10^5$.

~~(a)/(b) Nusselt number/RMS velocity vs. number of pressure and velocity DOF, at $Ra = 1 \times 10^5$, for a series of uniform, structured meshes in a 2-D cylindrical domain. High-resolution, adaptive mesh, results from the Fluidity computational modelling framework are delineated by dashed red lines; (c) final steady-state temperature field, with contours spanning temperatures of 0 to 1, at intervals of 0.05.~~

~~Convergence for 2-D cylindrical cases with zero-slip (a-d) and free-slip (e-h) boundary conditions, driven by smooth forcing at a series of different wave-numbers, n, and different polynomial orders of the radial dependence, k, as indicated in the legend (see **?**, for further details). Convergence rate is indicated by dashed lines, with the order of convergence provided in the legend.~~

```
# Mesh Generation:
rmin, rmax, ncells, nlayers = 1.22, 2.22, 256, 64
meshld = CircleManifoldMesh(ncells, radius=rmin, degree=2)
mesh = ExtrudedMesh(meshld, layers=nlayers, extrusion_type="radial")

-----------------------------------------------------------------------------------------
# Constants, unit vector, initial condition
Ra = Constant(1e5)
r = sqrt(X[0]**2 + X[1]**2)
k = as_vector((X[0], X[1])) / r
Told.interpolate(rmax-r + 0.02*cos(4.*atan_2(X[1],X[0]))*sin((r-rmin)*pi))

-----------------------------------------------------------------------------------------
# UFL for Stokes equations incorporating Nitsche:
C_ip = Constant(100.0) # Fudge factor for interior penalty term used in weak imposition of BCs
p_ip = 2 # Maximum polynomial degree of the _gradient_ of velocity

# Stokes equations in UFL form:
stress = 2 * mu * sym(grad(u))
F_stokes = inner(grad(v), stress) * dx - div(v) * p * dx + dot(n, v) * p * ds_tb - (dot(v, k) * Ra * Ttheta) * dx
F_stokes += -w * div(u) * dx + w * dot(n, u) * ds_tb  # Continuity equation

# nitsche free-slip BCs
F_stokes += -dot(v, n) * dot(dot(n, stress), n) * ds_tb
F_stokes += -dot(u, n) * dot(dot(n, 2 * mu * sym(grad(v))), n) * ds_tb
F_stokes += C_ip * mu * (p_ip + 1)**2 * FacetArea(mesh) / CellVolume(mesh) * dot(u, n) * dot(v, n) * ds_tb

-----------------------------------------------------------------------------------------
# Nullspaces and near-nullspaces:
x_rotV = Function(V).interpolate(as_vector((-X[1], X[0])))
V_nullspace = VectorSpaceBasis([x_rotV])
V_nullspace.orthonormalize()
p_nullspace = VectorSpaceBasis(constant=True) # Constant nullspace for pressure n
Z_nullspace = MixedVectorSpaceBasis(Z, [V_nullspace, p_nullspace]) # Setting mixed nullspace

# Generating near_nullspaces for GAMG:
nns_x  = Function(V).interpolate(Constant([1., 0.]))
nns_y  = Function(V).interpolate(Constant([0., 1.]))
V_near_nullspace = VectorSpaceBasis([nns_x, nns_y, x_rotV])
V_near_nullspace.orthonormalize()
Z_near_nullspace = MixedVectorSpaceBasis(Z, [V_near_nullspace, Z.sub(1)])

-----------------------------------------------------------------------------------------
# Updated solve calls:
stokes_problem = NonlinearVariationalProblem(F_stokes, z) # velocity BC's handled through Nitsche
stokes_solver = NonlinearVariationalSolver(stokes_problem, solver_parameters=stokes_solver_parameters, appctx={"
    mu": mu}, nullspace=Z_nullspace, transpose_nullspace=Z_nullspace, near_nullspace=Z_near_nullspace)
```

Listing 5. Difference in Firedrake code required to reproduce isoviscous case in a 2-D cylindrical shell domain.

~~For the cases plotted, the series of meshes start at refinement level 1, where the mesh consists of 1024 divisions in the tangential direction and 64 radial layers. At each subsequent level the mesh is refined by doubling resolution in both directions.~~

With a free-slip boundary condition on both boundaries, one can add an arbitrary rotation of the form $(-y, x) = r\hat{\theta}$ to the velocity solution (i.e. this case incorporates a velocity nullspace, as well as a pressure nullspace). As noted in Section 4, these lead to null-modes (eigenvectors) for the linear system, rendering the resulting matrix singular. In preconditioned Krylov methods these null-modes must be subtracted from the approximate solution at every iteration ~~(?)~~(e.g. ?), which we illustrate through this example. The key changes required to simulate this case, displayed in Listing 5, are:

1. Mesh generation: we generate a circular manifold mesh (with 256 elements in this example) and extrude in the radial direction, using the optional keyword argument `extrusion_type`, forming 64 layers (lines ~~2-5~~2-4). To better represent the curvature of the domain and ensure accuracy of our quadratic representation of velocity, we approximate the curved cylindrical shell domain quadratically, using the optional keyword argument `degree`$= 2$ (see Section 4 for further ~~detail~~details).

2. The unit vector, ~~k~~ $\hat{\boldsymbol{k}}$, points radially, in the direction opposite to gravity, as defined on line ~~11.~~ 10. The temperature field is initialised using Equation (41) on line ~~12.~~ 11.

3. Boundary conditions are no longer aligned with Cartesian directions. We use the Nitsche method (see Section 4.1) to impose our free-slip boundary conditions weakly (lines 15-27). The fudge factor in the interior penalty term is set to 100 on line 16, with Nitsche-related contributions to the UFL added on lines 24-27. Note that for extruded meshes in Firedrake, `ds_tb` denotes an integral over both the top and bottom surfaces of the mesh (`ds_t` and `ds_b` denote integrals over the top or bottom surface of the mesh, respectively). `FacetArea` and `CellVolume` return, respectively, A_f and V_{cf} required by Equation 17. Given that velocity boundary conditions are handled weakly through UFL, they are no longer passed to the Stokes problem as a separate option (line 46). Note that in addition to the Nitsche terms, the UFL for the Stokes equations now also includes boundary terms associated with the pressure gradient and velocity divergence terms, which were omitted in Cartesian cases (for details, see Section 4.1).

4. We define the rotational nullspace for velocity and combine this with the pressure nullspace in the mixed finite element space Z (lines ~~30-35~~30-34). Constant and rotational near-nullspaces, utilised by our GAMG preconditioner, are also defined on lines ~~37-42~~37-41, with this information passed to the solver on line ~~47.~~ 46. Note that iterative solver parameters, identical to those presented in the previous example, are used (see Section 5.3.1).

Our predicted Nusselt numbers and RMS velocities converge towards those of ~~Fluidity~~ existing codes with increasing resolution (Figure **??**), demonstrating the accuracy of our approach. ~~Predicted RMS velocities exceed those of Fluidity, albeit only by $\sim 0.1\%$, but lie within the bounds set by other codes for this case (Wilson, Pers. Comm., using TerraFERMA: ?).~~ To further assess the validity of our setup, we have confirmed the accuracy of our solutions to the Stokes system in this 2-D cylindrical shell geometry, through comparisons with analytical solutions from **?**, for both zero-slip and free-slip boundary conditions. These provide a suite of solutions based upon a smooth forcing term, at a range of wave-numbers n, with radial dependence

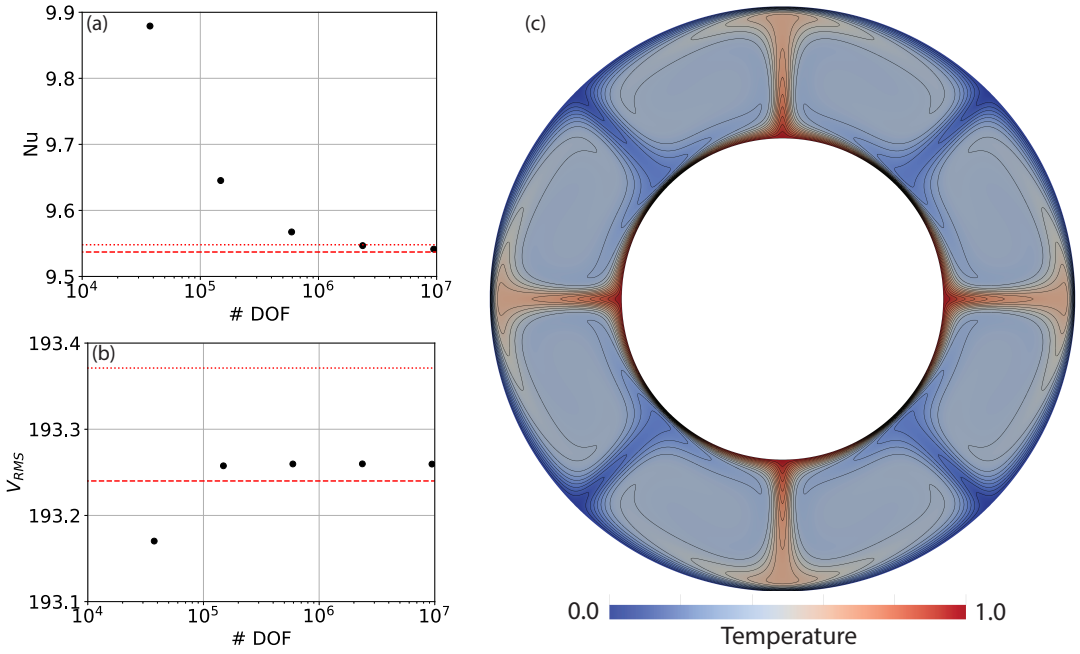

Figure 6. (a)/(b) Nusselt number/RMS velocity vs. number of pressure and velocity DOF, at $Ra = 1 \times 10^5$, for a series of uniform, structured meshes in a 2-D cylindrical shell domain. High-resolution, adaptive mesh, results from the Fluidity computational modelling framework (**?**) are delineated by dashed red lines, with results from ASPECT delineated by dotted red lines (**?**); (c) final steady-state temperature field, with contours spanning temperatures of 0 to 1, at intervals of 0.05.

formed by a polynomial of arbitrary order k. We study the convergence of our ~~Q2-Q1~~ Q2Q1 discretisation with respect to these solutions. Convergence plots are illustrated in Figure **??**. We observe super-convergence for the ~~Q2-Q1~~ Q2Q1 element pair at fourth- and second-order, for velocity and pressure, respectively, with both zero-slip and free-slip boundary conditions, which is higher than the theoretical (minimum) expected order of convergence of three for velocity and two for pressure (we note that super-convergence was also observed in **??**). Cases with lower wave-number, n, show smaller relative error than those at higher n, as expected. The same observation holds for lower and higher polynomial order, $k = 2$ and $k = 4$, for the radial density profile. To demonstrate the flexibility of Firedrake, we have also run comparisons against analytical solutions using a (discontinuous) delta-function forcing. In this case, convergence for the Q2Q1 discretisation (Figure **??**) drops to 1.5 and 0.5 for velocity and pressure, respectively. However, by employing the $Q2P_{1DG}$ finite element pair, we observe convergence at 3.5 and 2.0 (Figure **??**). Consistent with **?**, this demonstrates that the continuous approximation of pressure can lead to a reduced order of convergence in the presence of discontinuities, which can be overcome using a discontinuous pressure discretisation. Python scripts for these analytical comparisons can be found in the repository accompanying this paper.

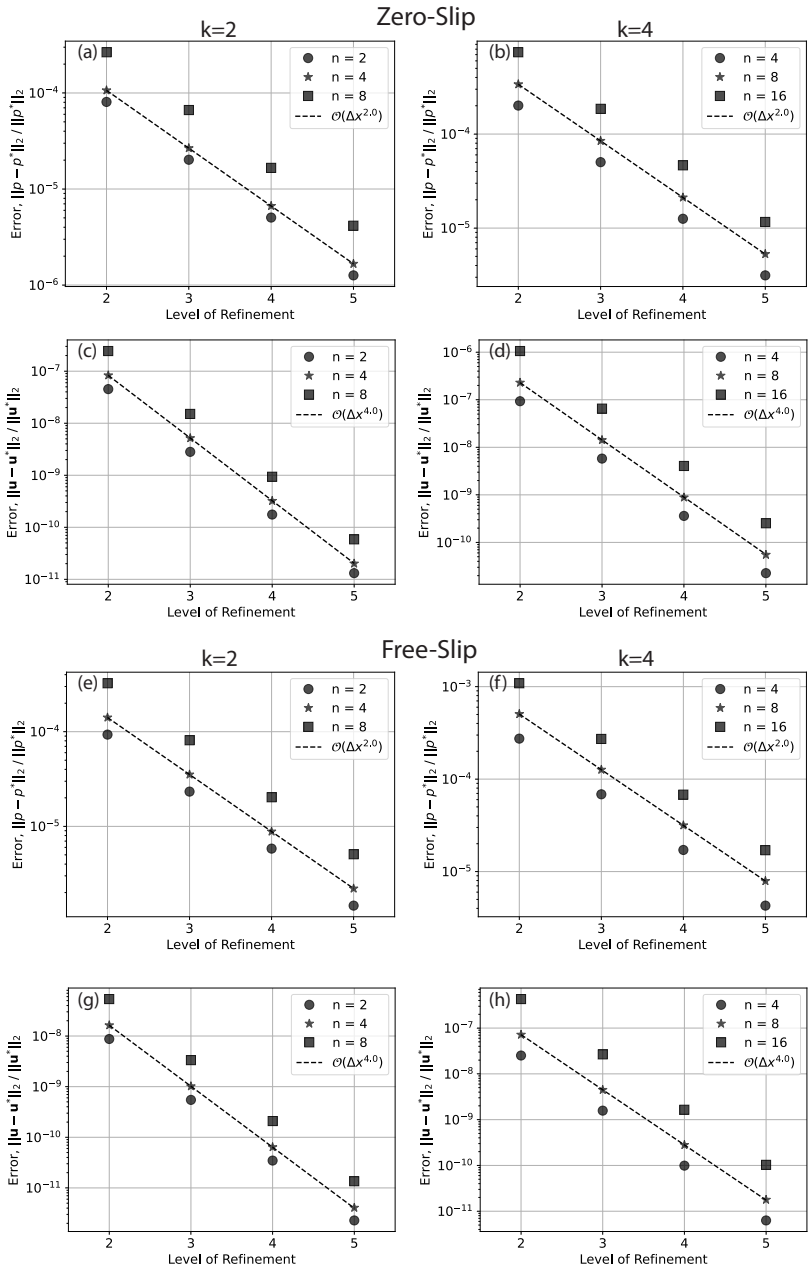

Figure 7. Convergence for 2-D cylindrical shell cases with zero-slip (a-d) and free-slip (e-h) boundary conditions, driven by smooth forcing at a series of different wave-numbers, n, and different polynomial orders of the radial dependence, k, as indicated in the legend (see **?**, for further details). Convergence rate is indicated by dashed lines, with the order of convergence provided in the legend. For the cases plotted, the series of meshes start at refinement level 1, where the mesh consists of 1024 divisions in the tangential direction and 64 radial layers. At each subsequent level the mesh is refined by doubling resolution in both directions.

```
# Mesh Generation:
rmin, rmax, ref_level, nlayers  = 1.22, 2.22, 4, 16
mesh2d = CubedSphereMesh(rmin, refinement_level=ref_level, degree=2)
mesh = ExtrudedMesh(mesh2d, layers=nlayers, extrusion_type='radial')

-----------------------------------------------------------------------------------------------
# Nullspaces and near-nullspaces:
x_rotV = Function(V).interpolate(as_vector((0, X[2], -X[1])))
y_rotV = Function(V).interpolate(as_vector((-X[2], 0, X[0])))
z_rotV = Function(V).interpolate(as_vector((-X[1], X[0], 0)))
V_nullspace = VectorSpaceBasis([x_rotV, y_rotV, z_rotV])
V_nullspace.orthonormalize()
p_nullspace = VectorSpaceBasis(constant=True) # Constant nullspace for pressure
Z_nullspace = MixedVectorSpaceBasis(Z, [V_nullspace, p_nullspace]) # Setting mixed nullspace

nns_x = Function(V).interpolate(Constant([1., 0., 0.]))
nns_y = Function(V).interpolate(Constant([0., 1., 0.]))
nns_z = Function(V).interpolate(Constant([0., 0., 1.]))
V_near_nullspace = VectorSpaceBasis([nns_x, nns_y, nns_z, x_rotV, y_rotV, z_rotV])
V_near_nullspace.orthonormalize()
Z_near_nullspace = MixedVectorSpaceBasis(Z, [V_near_nullspace, Z.sub(1)])
```

Listing 6. Difference in Firedrake code required to reproduce 3-D spherical shell benchmark cases from **?**.

5.3.3 3-D Spherical Shell Domain

We next move into a 3-D spherical shell geometry, which is required to simulate global mantle convection. We examine a well-known isoviscous community benchmark case (e.g. **????**), at a Rayleigh number of $Ra = 7 \times 10^3$, with free-slip velocity boundary conditions. Temperature boundary conditions are set to 1 at the base of the domain ($r_{\text{min}} = 1.22$) and 0 at the surface ($r_{\text{max}} = 2.22$), with the initial temperature distribution approximating a conductive profile with superimposed perturbations triggering tetrahedral symmetry at spherical harmonic degree $l = 3$ and order $m = 2$ (see **?**, for further details).

As illustrated in Listing 6, when compared to the 2-D cylindrical shell case examined in Section 5.3.2, the most notable change required to simulate this 3-D case is the generation of the underlying mesh. We use Firedrake's built-in `CubedSphereMesh` and extrude it radially through 16 layers, forming hexahedral elements. As with our cylindrical shell example, we approximate the curved ~~cylindrical~~ spherical domain quadratically, using the optional keyword argument `degree`$= 2$. Further required changes, highlighted in Listing 6, relate to 3-D extensions of the velocity nullspace, and the near-nullspaces required by the GAMG preconditioner, all of which are simple. We do not show the changes associated with extending the radial unit vector to 3-D, or the initial condition for temperature, given that they are straightforward, although, as with all examples, a complete Python script for this case can be found in the repository accompanying this paper.

Despite the simplicity of our setup, the accuracy of our approach is confirmed via comparison of both Nusselt numbers and RMS velocities with those of previous studies ~~(e.g. ????????)~~(e.g. **?????????**). For completeness, the final steady-state temperature field is illustrated in Figure **??**(c). Furthermore, in line with our 2-D cases, we have confirmed the accuracy of our Stokes solver for both zero-slip and free-slip boundary conditions in a 3-D spherical shell geometry, through comparisons with analytical solutions from **?**, which provide solutions based upon a smooth forcing term at a range of spherical harmonic degrees, l, and orders, m, with radial dependence formed by a polynomial of arbitrary order k. As with our 2-D cases, we observe superconvergence for the ~~Q2-Q1~~ Q2Q1 element pair at fourth- and second-order, for velocity and pressure, respectively, with both zero-slip and free-slip boundary conditions (Figure **??**).

This section has allowed us to highlight a number of Firedrake's benefits over other codes: (i) the ease at which simulations can be examined in different geometries, with minimal changes to the Python code, facilitated by Firedrake's built-in mesh

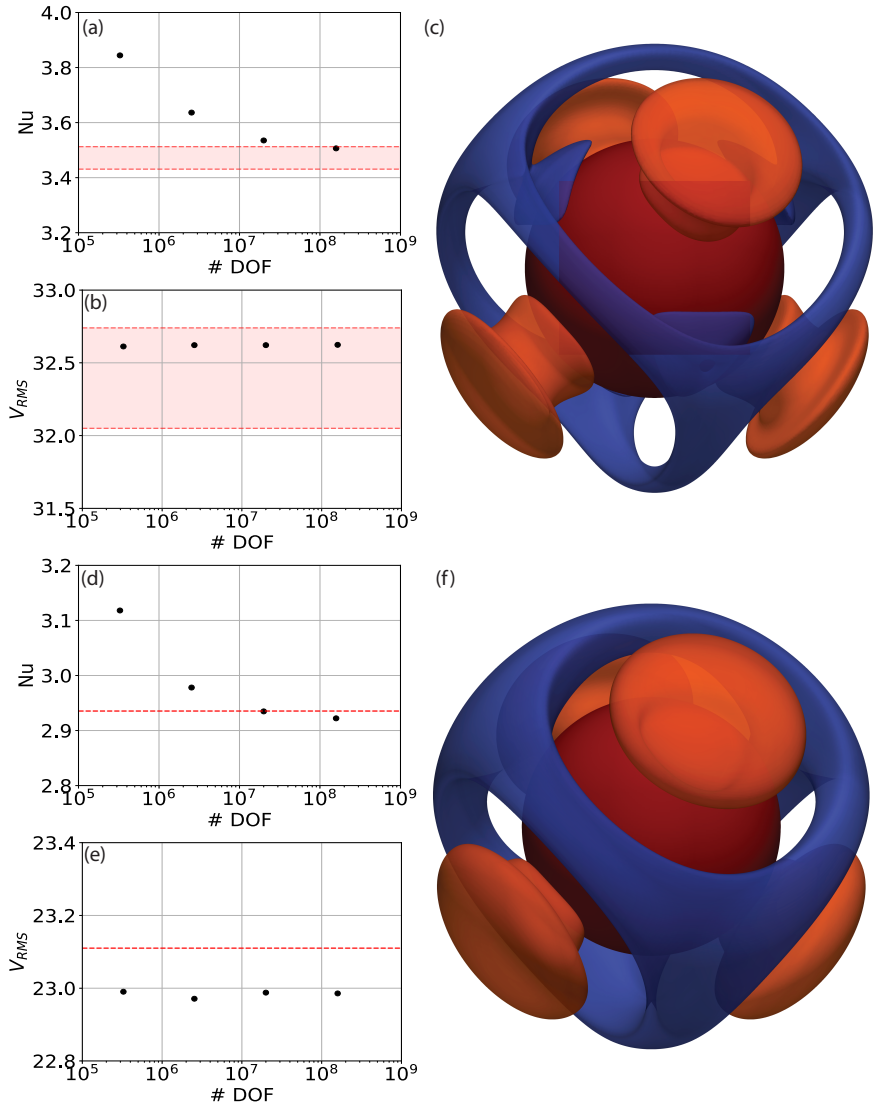

Figure 8. (a)/(b) Nusselt number/RMS velocity vs. number of pressure and velocity DOF, designed to match an isoviscous 3-D spherical shell benchmark case at $Ra = 7 \times 10^3$, for a series of uniform, structured meshes. The range of solutions predicted in previous studies are bounded by dashed red lines ~~(????????)~~(?????????); (c) final steady-state temperature field highlighted through isosurfaces at temperature anomalies (i.e. away from radial average) of $T = -0.15$ (blue) and $T = 0.15$ (orange), with the core-mantle-boundary at the base of the spherical shell marked by a red surface; (d-f) as in a-c, but for a temperature-dependent-viscosity case, with thermally induced viscosity contrasts of 10^2. Fewer codes have published predictions for this case, but results of **?** are marked by dashed red lines, for comparison.

generation utilities and extrusion functionality; (ii) the ease at which iterative solver configurations and preconditioners can be updated and tested, including scenarios incorporating multiple nullspaces, facilitated by Firedrake's fully-programmable solver interface, alongside its customisable preconditioner interface, both of which are seamlessly coupled to PETSc; and (iii)

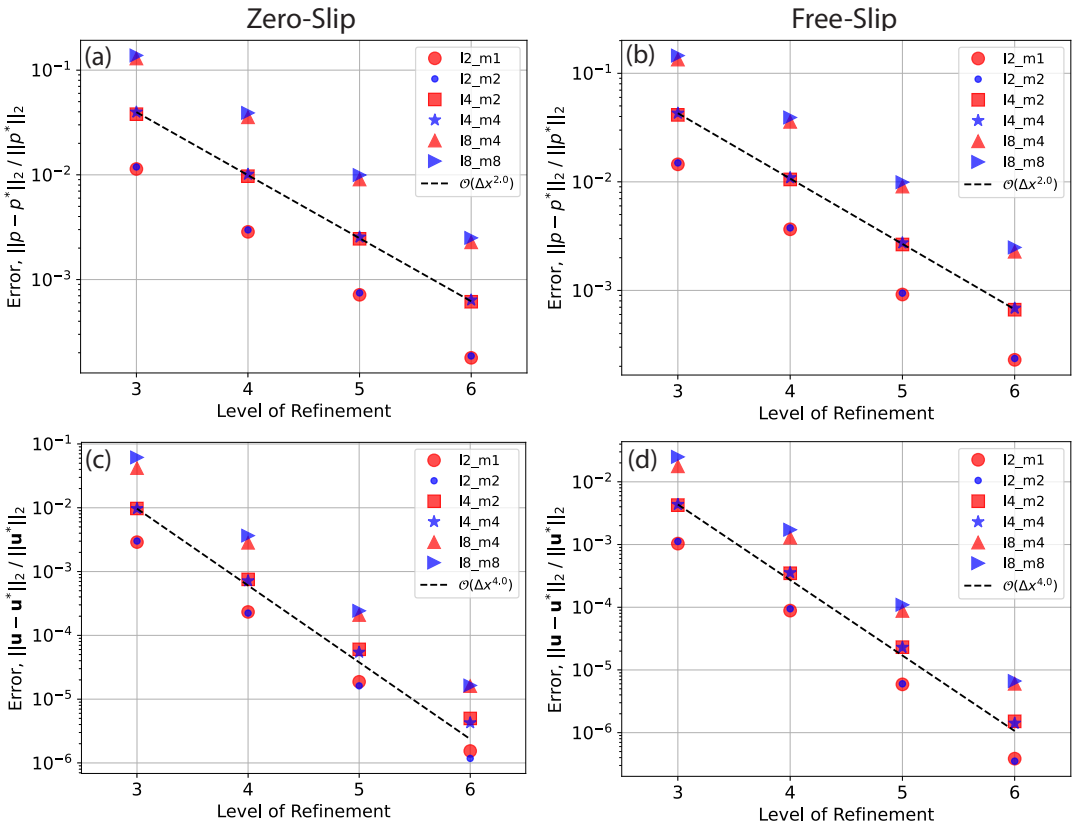

Figure 9. Convergence of velocity and pressure for 3-D spherical shell cases with zero-slip and free-slip boundary conditions, for perturbations at a range of spherical harmonic degrees l and orders m. Note that all cases with a smooth forcing are run at $k = l + 1$. Refinement level 3 corresponds to the level specified for our cubed sphere mesh, comprising 386 elements in the tangential direction, which is extruded radially to 8 layers. Resolution is doubled in all directions at subsequent refinement levels.

the convergence properties of our finite element system, in geometries that are representative of Earth's mantle. Taken together, these confirm Firedrake's suitability for simulations of global mantle dynamics, as will be further highlighted in Section **??**.

6 Parallel Scaling

We assess parallel scalability using a 3-D spherical shell case similar to that presented in Section **??**, albeit incorporating a temperature-dependent viscosity, following the relation:

$$\mu = \exp[E(0.5 - T)], \tag{42}$$

where E is a parameter that controls the temperature dependence of viscosity. In the example considered — Case A4 from **?** — we set $E = \ln(100)$, leading to thermally induced viscosity contrasts of 10^2 across the computational domain. For complete-

ness, our steady-state results, highlighting the consistency of our results for this case with the predictions of **?**, are displayed in Figure **??**, although for the purposes of parallel scaling analyses, we run simulations for 20 time-steps only.

We focus on weak scaling, where the problem size and the number of processing cores are simultaneously increased. Cases are examined on 24, 192, 1536 and 12288 cores, maintaining 4096 elements per core and ensuring a constant element aspect ratio across all resolutions examined. Simulations were examined on the Gadi supercomputer at the National Computational Infrastructure (NCI) in Australia, using compute nodes with 2×24 core Intel Xeon Platinum 8274 (Cascade Lake) 3.2 GHz CPUs, and 192 GB RAM, per node. Linking the nodes is the latest generation HDR InfiniBand technology in a Dragonfly+ topology, capable of transferring data at up to 200 Gb/s.

The most challenging aspect of weak parallel scaling is solver performance as the problem size increases. Whilst the amount of computation in equation assembly typically scales linearly with the number of DOFs – before taking parallel aspects such as communication into account – solver scaling is generally worse. In the case of iterative solvers, this is due to a deterioration in the conditioning of the matrix, driving an increase in the number of iterations required for convergence. As a result, even if the cost per iteration scales linearly, the overall cost will not. This implies that for weak scaling, the amount of work per core may increase rapidly, despite the number of DOFs per core remaining consistent.

The deterioration in conditioning is intimately related to the fact that an increase in resolution increases the ratio between smallest and largest resolvable length-scales. For elliptic operators, like the viscosity matrix K, the condition number scales with the square of that ratio (e.g. **?**). Multigrid approaches, which separate smaller and larger length scales on a hierarchy of fine to coarse meshes, are commonly used to address this problem, which motivates the choice of the algebraic multigrid preconditioner, GAMG, used here. Such approaches aim to maintain a constant, or only slowly increasing number of iterations and, thus, a near-linear scaling of the overall cost, as the problem size increases. This can be a challenge however as, for instance, an increase in resolution will require more multigrid levels, which will lead to an increased setup time and cost per iteration. In practice, when configuring the multigrid method, a compromise needs to be found between the effectiveness of multigrid in limiting the number of iterations, and not allowing the setup and costs per iteration to grow too rapidly. The two options, `gamg_threshold` and `gamg_square_graph`, specified in our solver setup, ensure a balance between multigrid effectiveness an coarse grid complexity.

A breakdown of key parallel scaling results are presented in Figure **??**. Panel ~~a~~ (a) displays the average number of iterations per solve over the 20 timesteps. We find that the number of pressure (the Schur complement solve: `fieldsplit_1`) and energy solve iterations remains flat (12 and ~ 10.5, respectively), whilst the number of velocity solve iterations (inversion of the matrix K, using the GAMG preconditioner: `fieldsplit_0`) increases only slowly, from ~ 41 to ~ 51, over a greater than three-orders-of-magnitude increase in problem size and number of processor cores. This demonstrates algorithmic scalability on up to 12288 cores and $\sim 50 \times 10^6$ elements (which corresponds to ~~$\sim 1.25 \times 10^9$~~ $\sim 1.26 \times 10^9$ velocity and pressure degrees of freedom).

Parallel scalability can also be assessed by analysing the growth in CPU-time of the dominant components of our problem: assembly of finite element systems (Figure **??**b), setup of the algebraic multigrid (GAMG) preconditioner (Figure **??**c), and time spent solving the Schur complement system (Figure **??**d). We find that the assembly time is a negligible fraction of

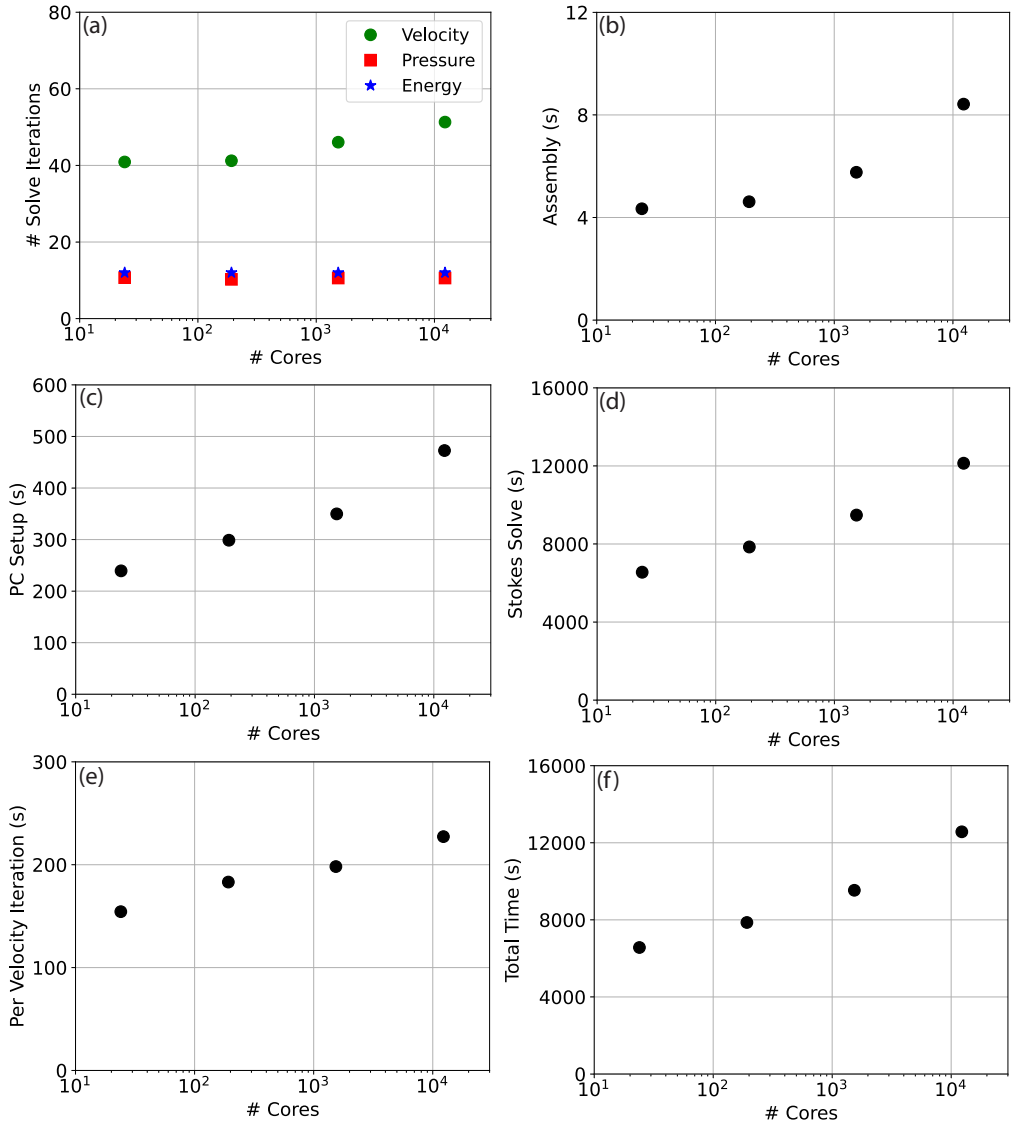

Figure 10. Weak scaling analyses for a 20 time-step, temperature-dependent viscosity, spherical shell simulation with free-slip boundary conditions: (a) mean number of iterations per time-step, for energy (blue stars), pressure (red squares) and velocity (green circles) solves, respectively; (b) time spent in assembly of finite element systems; (c) time spent setting up algebraic ~~multgird~~ multigrid preconditioner; (d) time spent solving the Schur complement (Stokes) system; (e) cost per velocity solve iterations; (f) total simulation time, which closely mimics the Schur complement solution time.

this problem. The setup time for our GAMG preconditioner grows from ~ 240s on 24 cores to ~ 470s on 12288 cores. This is understandable, given the large communication costs associated with setting up various multigrid levels, particularly for problems incorporating nullspaces and near-nullspaces, as is the case here. We note, however, that this is not a concern: as a

fraction of the entire solution time for the Schur complement solve (Figure **??**d), GAMG setup remains small. We do observe an increase in time required for solution of the Schur Complement (Stokes solve), from ~ 6500s on 24 cores to ~ 12100s on 12288 cores. This results primarily from the minor increase in the number of velocity solve iterations and the increased cost per iteration (Figure **??**e), which rises from 155s on 24 cores to 225s on 12288 cores, reflecting costs associated with increasing the number of multigrid levels for higher-resolution problems. The total time spent in running this problem mirrors the time spent in solving the Schur complement system (Figure **??**f), indicating where future optimisation efforts should be directed. ~~We note that the change in gradient, apparent in panels b-f when moving from 24-192 and 192-12288 cores arises due to a transition from running simulations on a single compute node to multiple nodes.~~

7 ~~Realistic~~ Application in 3-D spherical shell geometry: Global Mantle Convection

In this section, we demonstrate application of Firedrake to a time-dependent simulation of global mantle convection in a 3-D spherical shell geometry, at a realistic Rayleigh number. We assume a compressible mantle, under the Anelastic Liquid Approximation (ALA), and a temperature-, depth-, and strain-rate dependent rheology, in line with the viscoplastic case analysed in Section 5.2.2. Viscosity increases below the mantle transition zone and we include a brittle-failure type yield-stress law, ensuring that yielding concentrates at shallow depths. As with the examples provided above, calculations are performed using a hexahedral ~~trilinear Q2-Q1~~ Q2Q1 element pair for velocity and pressure. We use a Q2 discretisation for temperature and, given the increased importance of advection at higher Rayleigh numbers, incorporate stabilisation through a streamline upwinding scheme, following **?**. We employ a Cubed Sphere mesh with 98304 elements on each spherical surface and extrude it radially through 64 layers, with spacing reduced adjacent to the top and bottom boundaries of the domain. This results in a problem with $\sim 1.26 \times 10^9$ velocity and pressure degrees of freedom, and $\sim 5.0 \times 10^7$ temperature degrees of freedom. Our solution strategy for the Stokes ~~and energy equations is otherwise identical~~ equations is similar to the spherical shell examples presented above~~.~~, albeit exploiting PETSc's SNES functionality using a setup based on Newton's method to handle the nonlinearity in the system. Solution strategy for the energy equation is identical to the previous example.

~~For simplicity, we assume an incompressible mantle and a linear temperature- and depth-dependent rheology, following the relation,~~

$$\mu = \mu_0 \exp[E(0.5 - T)]\,.$$

~~Here μ_0 is a reference viscosity that increases by a factor of 40 below the mantle transition zone, and $E = \ln(1000)$ controls the sensitivity of viscosity to temperature. We specify a reference~~ We achieve a (basally heated) Rayleigh number of ~~2×10^7~~ 7.5×10^7 in the asthenosphere, which is comparable to estimates of Earth's mantle (e.g. **?**), and also include internal heating at a non-dimensional heating rate of 10. The simulation is spun-up with free-slip and isothermal ~~($T = 0$ at base; $T = 1$ at top)~~ boundaries at both surfaces. After the model reaches a quasi-steady state (i.e. when the surface and basal Nusselt numbers both change by less than 0.1% over 10 consecutive time-steps), surface velocities are assimilated through a kinematic boundary condition, according to 230 Myr of plate motion histories (**?**), using the Python interface to GPlates (e.g. **??**). Our simulation then runs for-

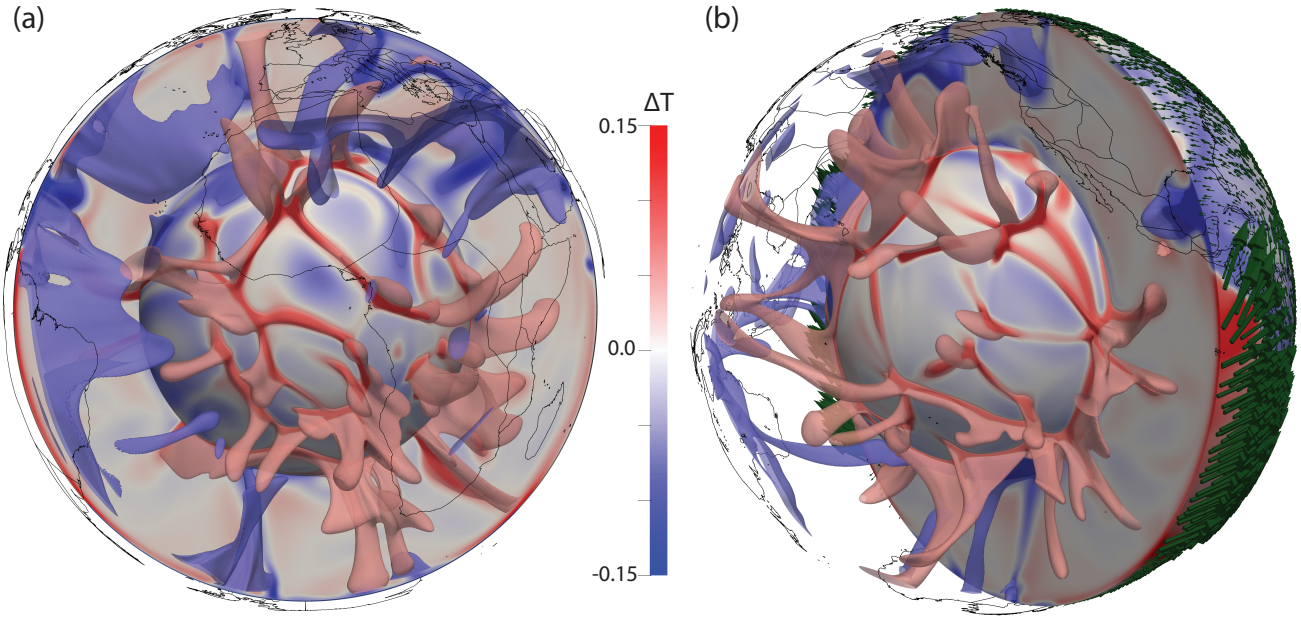

Figure 11. Present-day thermal structure, predicted from our global mantle convection simulation where the geographic distribution of heterogeneity is dictated by 230 Myr of imposed plate motion history (**?**). Each image includes a radial surface at $r = 1.25$ (i.e. immediately above the core-mantle boundary), a cross-section, and transparent isosurfaces at temperature anomalies (i.e. away from the radial average) of ~~$T = -0.15$~~ $T = -0.075$ (blue) and ~~$T = 0.15$~~ $T = 0.075$ (red), highlighting the location of downwelling slabs and upwelling mantle plumes (below ~~$r = 2.19$~~$r = 2.13$), respectively. Continental boundaries provide geographic reference. Panel a provides an Africa-centered view, with panel b centered on the Pacific Ocean, and including (green) glyphs at the surface highlighting the imposed plate velocities.

ward towards the ~~present-day~~present day. This case is therefore ~~analogous~~ similar to the simulations examined when addressing questions from the very frontiers of geodynamical research~~(e.g. ?????????).~~ , albeit incorporating a more representative treatment of mantle and lithosphere rheology (e.g. **??????????**). The simulation was executed across 1344 CPUs, on the same architecture as outlined above, and took ~ 92 hours.

Our results are illustrated in Figure **??**. We find that the present-day upper mantle convective planform is dominated by strong downwellings in regions of plate convergence. In the mid-mantle, cold downwellings are prominent beneath North America and South-East Asia, whilst remnants of older subduction are visible above the core-mantle-boundary. The location of hot upwelling material is strongly modulated by these downwellings, with upwelling plumes concentrating in two clusters beneath the African continent and the central Pacific ocean (i.e. away from regions that have experienced subduction over the past 150 Myr or so). The cluster of plumes in the Pacific is reasonably circular, whilst those beneath Africa extend in a NW-SE trending structure, which to the north curves eastward under Europe and to the south extends into the Indian Ocean.

Further analysis of this proof-of-concept simulation is beyond the scope of this study. However, when combined with the benchmark and parallel scaling analyses presented-above, our model predictions, which are consistent with those from a num-

ber of previous studies (e.g. **????**), confirm Firedrake's applicability for ~~realistic,~~ time-dependent ~~,~~ global mantle dynamics simulations of this nature.

8 Discussion

Firedrake is a next-generation automated system for solving variational problems using the finite element method (e.g. **??**). It has a number of features that are ideally suited to simulating geophysical fluid dynamics problems, as exemplified by its use in application areas such as coastal ocean modelling (**?**), numerical weather prediction (**?**), and glacier flow modelling (**?**). The focus of this manuscript has been to demonstrate Firedrake's applicability for geodynamical simulation, with an emphasis on global mantle dynamics. To do so, we have presented, analysed and validated Firedrake against a number of benchmark and analytical cases ~~,~~ of systematically increasing complexity, building towards a ~~realistic~~ time-dependent global simulation at realistic convective vigour.

~~In order to introduce the~~ To introduce its core components and illustrate the elegance of setting up and validating a geodynamical model in Firedrake, we started with a simple, incompressible, isoviscous case in an enclosed 2-D Cartesian box. Setting up this problem was straightforward, requiring only a weak formulation of the governing equations for specification in UFL, together with a mesh, ~~boundary conditions~~ initial and boundary conditions, and appropriate discrete function spaces. ~~By utilising~~ We utilised Firedrake's built-in meshing functionality and default direct solver options, ~~we were able to demonstrate~~ and demonstrated the framework's accuracy for simulations of this nature: in ~~less than 70~~ only 56 lines of Python (excluding comments and blank lines), we reproduced results from the well-established benchmark study of **?**~~,~~.

Representative simulations of mantle and lithosphere dynamics, however, incorporate more complicated physics. To demonstrate Firedrake's applicability in such scenarios, we next set up 2-D simulations that accounted for compressibility, through the Anelastic Liquid Approximation (**?**), and a nonlinear viscosity that depends upon temperature, depth and strain-rate. Our results were validated through comparison with the benchmark studies of ~~**?**~~ **?** and **?**, respectively. For compressible cases, despite the governing equations differing appreciably from their incompressible counterparts, the modifications required to our setup were minimal, with the most notable change being the UFL describing the relevant PDEs. For the viscoplastic rheology case, where viscosity varied by several orders of magnitude across the domain, an appropriate solution strategy was required to deal with nonlinear coupling between strain-rate and viscosity: Firedrake's fully-programmable solver interface and seamless coupling to PETSc facilitated the straightforward use of PETSc's Scalable Nonlinear Equation Solvers (SNES) (**?**). Taken together, these examples highlight one of Firedrake's key benefits: by leveraging UFL (**?**), associated strategies for automatic assembly of finite element systems, and PETSc (**???**), the framework is easily extensible, allowing for straightforward application to problems involving different physical approximations, even when they require distinct solution strategies.

This is further highlighted with the transition from 2-D to 3-D. With modifications to only a few lines of Python, the basic 2-D Cartesian case described above was easily extended to 3-D, allowing for comparison and validation against the well-established benchmark results of **?**. However, the direct solvers used for our 2-D cases quickly become computationally intractable in 3-D, necessitating the use of an iterative approach. Firedrake's programmable solver interface facilitates the

straightforward inclusion of Python dictionaries that define iterative solver parameters for the Stokes and energy systems. A number of different schemes have been advocated by the geodynamical modelling community (e.g. **??**), but in all 3-D simulations examined herein, the Schur complement approach was utilised for solution of our Stokes system, exploiting the fieldsplit preconditioner type to apply preconditioners, including algebraic multigrid, to different blocks of the system. A ~~Crank-Nicholson~~ Crank-Nicolson scheme was utilised for temporal discretisation of the energy equation, with a standard GMRES Krylov method with SOR preconditioning used for solution. We have demonstrated that such solution strategies are effective and scalable, with algorithmic scalability confirmed on up to 12288 cores.

Cartesian simulations offer a means to better understand the physical mechanisms controlling mantle convection, but a 3-D spherical shell geometry is required to simulate global mantle dynamics. We have demonstrated how Firedrake's built-in meshing and extrusion functionality facilitates the effortless transition to such geometries (in addition to comparable 2-D cylindrical shell geometries), whilst its Python user-interface allows for the simple inclusion of a radial gravity direction and boundary conditions that are not aligned with Cartesian directions. The convergence properties and accuracy of our simulations in a 3-D spherical shell geometry have been demonstrated through comparison with the extensive set of analytical solutions introduced by **?** and a series of low Rayleigh number isoviscous and temperature-dependent viscosity simulations, from **?**. We observed super-convergence for the ~~Q2-Q1~~ Q2Q1 element pair at fourth- and second-order, for velocity and pressure, respectively.

Having validated Firedrake against this broad suite of cases, we finally applied the framework to a ~~realistic~~ simulation of global mantle convection ~~. For simplicity, we assumed an incompressible~~ at realistic convective vigour. We assumed a compressible mantle and a ~~linear temperature- and depth-dependent rheology~~nonlinear temperature, depth and strain-rate dependent viscosity, assimilating 230 Myr of plate motion histories (**?**) through a kinematic surface boundary condition. These prescribed plate velocities ~~organize~~ modulate underlying mantle flow, such that the predicted present-day convective planform is dominated by cold downwellings in regions of plate convergence, with upwellings concentrating elsewhere, particularly beneath the African ~~and Pacific domains~~continent and Pacific Ocean. Our model predictions, which are consistent with those from a number of previous studies (e.g. **?????**), reproduce first-order characteristics of the structure of Earth's mantle imaged through seismology (e.g. **??**) ~~,~~ and the geographical distribution of mantle plumes (e.g. **??**)~~, and are consistent with those from a number of previous studies and the (e.g. **????**),~~ . They serve as a proof-of-concept, confirming Firedrake's applicability for ~~realistic,~~ time-dependent, global simulations of this nature and, accordingly, its suitability for addressing research problems from the very frontiers of global geodynamical research.

Despite this, several components of Firedrake have not been fully examined in this paper. Many of these will likely be useful for geodynamical simulation and, accordingly, will be examined in the future. These include:

1. A range of finite elements: in ~~all~~ most examples considered herein, we utilised a continuous ~~Q2-Q1~~ Q2Q1 element pair for velocity and pressure with a Q2 discretisation for temperature~~(with the exception of one set of examples in Section 5.1, where we demonstrated the use of~~ . In addition, in some cases we instead employ the $Q2P_{1DG}$ finite element pair for the Stokes system, or a Q1 ~~temperature discretisation). Accordingly~~discretisation for temperature. Despite this, we have not fully demonstrated Firedrake's support for a ~~wide-range~~ wide-array of finite elements, including continuous,

```
1 class Mass(AuxiliaryOperatorPC):
2     def form(self, pc, test, trial):
3         a = 1/mu * inner(test, trial)*dx
4         bcs = None
5         return (a, bcs)
```

Listing 7. Re-implementation by user code of the `MassInvPC` preconditioner for the Schur complement first used in Section 5.3.1. The UFL in line 3 that defines a mass matrix scaled by the inverse of viscosity μ could be replaced by any other expression approximating the Schur complement (see **?** for an overview). Preconditioners that are expressed through linear algebra operations on sub-matrices of the saddle point matrix, e.g. $G^T K G \approx \mathrm{diag}\left(G^T \mathrm{diag}(K) G^T\right)$, can be constructed by applying these operations through the `petsc4py` interface.

discontinuous, H(div) and H(curl) discretisations, and elements with continuous derivatives such as the Argyris and Bell elements (see **?**, for an overview). Some of these could offer major advantages for geodynamical simulation. ~~For example,~~

2. The use of DG schemes for the solution of the energy equation: a number of studies now advocate the use of Discontinuous Galerkin (DG) schemes for solution of the energy equation (e.g. **??**). Importantly, Firedrake's simple API allows a user to escape the UFL abstraction, and implement common operations that fall outside of pure variational formulations, such as flux limiters, which are central to DG schemes. ~~Firedrake also~~

3. Hybridisation strategies: Firedrake provides the necessary infrastructure for hybridisation strategies (**?**), which allow for a reduction of the many extra degrees of freedom introduced by DG schemes in the global system to a smaller subset, defined on element interfaces through so-called trace elements. This ~~offers the prospect of arriving at~~ could facilitate more efficient ways of solving the Stokes system (e.g. **?**). Such possibilities will be explored in future work, noting that Firedrake's existing support for these elements will facilitate rapid and efficient testing and validation.

4. Fully coupled nonlinear systems: in all examples considered herein, we solve for velocity and pressure in a separate step to temperature, largely owing to our familiarity with this approach from previous work ~~(e.g. **??**)~~(e.g. **???**). However, a number of studies advocate solving for these fields simultaneously (e.g. **?**), particularly for strongly coupled, highly-nonlinear, multi-physics problems. By leveraging UFL, in combination with PETSc's fieldsplit preconditioning approach, future work to configure and test such coupled schemes within Firedrake will be relatively straightforward.

5. Preconditioners: a major benefit of Firedrake for the problems considered herein is access to the wide variety of solution algorithms and preconditioning strategies provided by the PETSc library, which can be flexibly configured through the solver parameters dictionary, allowing one to test and apply different strategies with ease. The development of preconditioners for the Stokes problem is an active area of research ~~(e.g. **???**)~~(e.g. **????**). As noted above, Firedrake supports a powerful programmable preconditioner interface which, in turn, connects with the Python preconditioner interface of PETSc, and allows users to specify their own linear operator in UFL ~~, thus~~ (see Listing **??** for an example) enabling preconditioning techniques with bespoke operator approximations. We note that in addition to the complete range of algebraic solvers offered by PETSc, Firedrake also provides access to multilevel solvers with geometric hierarchies, opening up the possibility of exploring geometric multigrid approaches in the future.

To support these statements and further demonstrate the potential of the framework in exploring challenging nonlinear problems, we briefly consider the nonlinear benchmark case of **?**, with a strain-rate and pressure dependent Drucker-Prager rheology. In **?**, a number of solution strategies are explored for this case (amongst others), with the study advocating the use of two modifications to the Jacobian: (i) adding an additional term to Equation (32) that is the transpose of the second term, thus restoring the symmetry of K; (ii) to scale those terms associated with $\partial\eta/\partial\boldsymbol{u}$ by a spatially varying α_{SPD}, calculated at the Gauss points according to:

$$\alpha_{\text{SPD}} = \begin{cases} 1 & \text{if } \left[1 - \frac{a:b}{\|a\|\|b\|}\right]^2 < c_{\text{safety}} 2\eta(\dot{\epsilon}(\boldsymbol{u})), \\ c_{\text{safety}} \frac{2\eta(\dot{\epsilon}(\boldsymbol{u}))}{\left[1 - \frac{a:b}{\|a\|\|b\|}\right]^2} & \text{otherwise,} \end{cases} \tag{43}$$

where $a = \dot{\epsilon}$, and $b = \frac{\partial\eta}{\partial\dot{\epsilon}}$. This rescaling acts as a stabilisation, ensuring that K remains positive definite. It should be noted that the pressure dependence of the Drucker-Prager rheology also leads to additional terms in the top-right block of the Stokes Jacobian matrix, in addition to G in Equation (28), making the overall system asymmetric, regardless.

In traditional codes, the implementation of such additional terms in the Jacobian and the proposed modifications (stabilisation) require significant development. Analytical expressions for $\partial\eta/\partial\boldsymbol{u}$ and $\partial\eta/\partial p$ must be derived for each specific rheological relationship analysed (as is done in the appendices of **?**), and the assembly of any additional terms may require a significant overhaul of existing code and data structures as, for example, sparsity structures may change. In Firedrake, the full Jacobian is derived symbolically and the code for its assembly generated automatically, making the entire process automatic, even for highly complex rheologies. We were able to implement the Jacobian modifications proposed in **?** in only 7 lines of Python code (the full Python script for this case is available in the repository accompanying this paper) and, as illustrated in Figure **??**, we obtain similar results. As indicated in **?**, the convergence of the problem gets more challenging with increased resolution, and although a reasonably converged result can be obtained for the case shown in Figure **??** at a resolution of 1024×512, this is insufficient to resolve the details of the unstructured mesh domain used in **?** who reported non-convergence for this case. Firedrake's ability to choose from a large variety of discretisation types, including unstructured meshes, and its flexibility to adapt and experiment with the solution strategy, opens up numerous avenues to further investigate the challenges in this, and other, highly nonlinear problems.

It is important to point out that some common components of geodynamical models have not been showcased herein and, to our knowledge, have not yet been explored within the Firedrake framework. These include, for example, a free-surface boundary condition and the ability to model multiple-material flows, often implemented in geodynamical models using the particle-in-cell technique. Our goal for this paper is to provide solid foundations for future work in Firedrake that we, and others in the geodynamical modelling community, can build upon. Nonetheless, we see no fundamental reason why any component of other geodynamical modelling tools cannot be incorporated within Firedrake. For example, the TerraFERMA framework of **?**, which is built on FEniCS, has been able to match the free-surface benchmarks of **?** – a similar implementation would be straightforward in Firedrake. For multi-material flows, solving an advection equation, for example with a Discontinuous Galerkin scheme and appropriate limiters (e.g. **?**), would be straightforward. In addition, particle-in-cell schemes have been

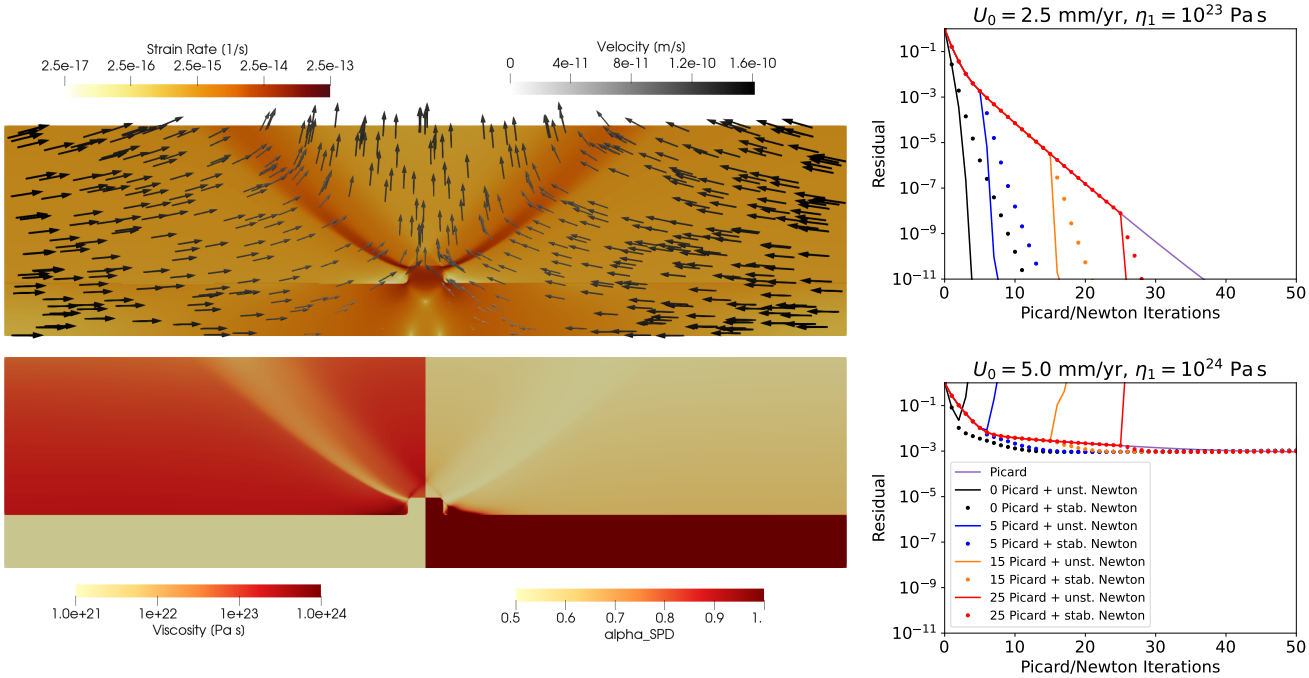

Figure 12. Benchmark case of **?** with strain-rate and pressure dependent Drucker-Prager rheology. Solution fields, including velocity, strain rate, viscosity, and α_{SPD} (see Equation **??**) are shown for the case with inflow velocity $U_0 = 5$ mm/yr, $\eta_1 = 10^{24}$ Pa s and a friction angle of $\alpha = 30^o$ in the left panel. The top-right and bottom-right panel show convergence of the residual in the Picard and Newton solvers applied to the $U_0 = 2.5$ mm/yr, $\eta_1 = 10^{23}$ Pa s case, and the $U_0 = 5$ mm/yr, $\eta_1 = 10^{24}$ Pa s case respectively. Both cases are run with a number of initial Picard iterations, as indicated in the legend, before switching to either the full unmodified Newton method (solid lines), or the stabilised method with modifications as proposed in **?** (dots). In the former case, the unmodified Newton method clearly performs best, with the stabilised method showing some degradation towards the Picard method (purple line). In the latter case, the unmodified Newton method fails to converge, whereas the stabilised method continues to converge slowly but not much faster than the Picard method, before stalling altogether. These results are generally consistent with those in **?**.

successfully developed and tested with FEniCS (**?**) and we see no fundamental reason that such functionality cannot be incorporated within Firedrake. Finally, Firedrake's flexibility would make exploring different advection schemes straightforward, rendering it very well-suited to level-set approaches (e.g. **?**).

We note that the automated approach underpinning Firedrake has the potential to revolutionize the use of adjoints and other inverse schemes in geodynamics. Adjoint models have made an enormous impact in fields such as meteorology and oceanography. However, despite significant progress (e.g. **??????**), their use in other scientific fields, including geodynamics, has been hampered by the practical difficulty of their derivation and implementation. In contrast to developing a model directly in Fortran or C++, high-level systems, such as Firedrake, allow the developer to express the variational problems to be solved in near-mathematical notation through UFL. As such, these systems have a key advantage: since the mathematical structure of

the problem is preserved, they are more amenable to automated analysis and manipulation, which can be exploited to automate the derivation of adjoints (e.g. **??**) and the generation of the low-level code for the derived models. Exploring the use of such an approach in geodynamics will be an important avenue for future research.

Finally, given ~~that~~ the importance of reproducibility in the computational geosciences~~is increasingly being recognized~~, we note that Firedrake integrates with Zenodo and GitHub to provide users with the ability to generate a set of DOIs corresponding to the exact set of Firedrake components used to conduct a particular set of simulations. In providing our input scripts and a DOI for the version of Firedrake used herein, we ensure traceable provenance of model data, in full compliance with FAIR (Findable, Accessible, Interoperable, Reusable) principles.

9 Conclusions

Firedrake is a next-generation system for solving variational problems using the finite element method (e.g. **??**). It treats finite element problems as a composition of several abstract processes, using separate and open-source software components for each. Firedrake's overarching goal is to save users from manually writing low-level code for assembling the systems of equations that discretize their model physics. It is written completely in Python, and exploits automatic code-generation techniques to apply sophisticated performance optimisations. Firedrake creates a separation of concerns between employing the finite element method and implementing it: this is a game-changer, as it opens up these problems to a new class of user and developer.

In this manuscript, we have confirmed Firedrake's applicability for geodynamical simulation, by configuring and validating model predictions against a series of benchmark and analytical cases, of systematically increasing complexity. In all cases, Firedrake has been shown to be *accurate* and *efficient*, and we have also demonstrated that that it is *flexible* and easily *extensible*: by leveraging UFL and PETSc, it can be effortlessly applied to problems involving different physical approximations (e.g. incompressible and compressible flow; isoviscous and more complex nonlinear rheologies), even if they require distinct solution strategies. We have illustrated how Firedrake's built-in mesh generation utilities and extrusion functionality provide a straightforward mechanism for examining problems in different geometries (2-D and 3-D Cartesian, 2-D cylindrical and 3-D spherical shells), and how its fully-programmable solver dictionary and customisable preconditioner interface, both of which are seamlessly coupled to PETSc, facilitate straightforward configuration of different solution approaches. Parallel *scalability* has been demonstrated, on up to 12288 compute cores. Finally, using a ~~realistic~~ more-representative simulation of global mantle dynamics, where the distribution of heterogeneity is governed by imposed plate motion histories (**?**), we have confirmed Firedrake's suitability for tackling challenges ~~from~~ at the very forefront of geodynamical research. We note that all simulation data presented herein has traceable provenance: in providing our input scripts and a DOI for the exact set of Firedrake components employed, Firedrake facilitates *transparency* and *reproducibility*, in full compliance with FAIR principles.

Code and data availability. Minor adjustments to the Firedrake code base required to successfully run the cases in this paper have been merged into the open-source software associated with the Firedrake Project: https://www.firedrakeproject.org/. For the specific components of

the Firedrake project used in this paper, see https://zenodo.org/record/6522930. For the input files of all examples and benchmarks presented, see https://zenodo.org/record/6523264.

Author contributions. DRD and SCK conceived this study, with all authors having significant input on the design, development and validation of the examples and cases presented. All authors contributed towards writing the manuscript.

Competing interests. Authors declare that they have no conflict of interest.

Acknowledgements. All authors acknowledge support from the Australian Research Data Commons (ARDC: https://ardc.edu.au/, under the G-Adopt platform grant: PL031), AuScope, Geosciences Australia and the National Computational Infrastructure (NCI). DRD and SCK acknowledge support from the Australian Research Council, under grant no. DP170100058. Numerical simulations were undertaken at the NCI National Facility in Canberra, Australia, which is supported by the Australian Commonwealth Government. Authors are grateful to the entire Firedrake development team, particularly David Ham, for support and advice at various points of this research. We are also grateful to 7 reviewers, including Cedric Thieulot, Marcus Mohr, Wolfgang Bangerth and Carsten Burstedde: their careful and constructive feedback helped to clarify and improve this contribution.

Appendix A: Governing Equations under the Anelastic Liquid Approximation

Density changes across Earth's mantle result primarily from hydrostatic compression, with density increasing by $\approx 65\%$ from surface to core-mantle-boundary (CMB) (e.g. **?**). Variations in density associated with local temperature and pressure perturbations are small in comparison to the spherically averaged density. For a chemically homogeneous mantle, it is therefore appropriate to assume a linearized equation of state, of the form:

$$\begin{aligned} \rho &= \bar{\rho}(\bar{T},\bar{p}) + \rho', \\ &= \bar{\rho}(\bar{T},\bar{p}) + \bar{\rho}(\bar{\chi}_T p' - \bar{\alpha} T'). \end{aligned} \tag{A1}$$

Here ρ, p, T, χ_T and α denote density, pressure, temperature, isothermal compressibility and the coefficient of thermal expansion, respectively, whilst overbars refer to a reference state and primes to departures from it:

$$T = \bar{T} + T', \qquad p = \bar{p} + p'. \tag{A2}$$

It is convenient to take the reference state as motionless and steady. Accordingly, for the purposes of the compressible case examined herein, we will assume that the reference state varies as a function of depth, z, only. The reference state pressure thus satisfies the hydrostatic approximation:

$$\frac{\partial \bar{p}}{\partial z} = \bar{\rho} g \cdot \hat{\boldsymbol{k}}, \tag{A3}$$

where $\mathbf{g}$ is the acceleration of gravity and $\hat{\boldsymbol{k}}$ is the unit vector in the direction opposite to gravity. On Earth, $\mathbf{g}$ is a function of position, however, for simplicity, it will be assumed constant for the compressible case examined herein. Following ??, the reference density and reference temperature are described through an adiabatic Adams–Williamson equation of state (**?**), where:

$$\bar{\rho}(z) = \rho_0 \exp\left(\frac{\alpha_0 g_0}{\gamma_0 c_{p_0}} z\right) \tag{A4}$$

and:

$$\bar{T}(z) = T_s \exp\left(\frac{\alpha_0 g_0}{c_{p_0}} z\right). \tag{A5}$$

Here, c_p and T_s represent the specific heat capacity at constant pressure and surface temperature, respectively, whilst γ_0 denotes the Grüneisen parameter, given by:

$$\gamma_0 = \frac{\alpha_0}{\rho_0 c_{v_0} \chi_{T_0}}, \tag{A6}$$

where c_v denotes the specific heat capacity at constant volume. Variables with a sub-script 0 are constants, used in defining the reference state. Here, they are defined at the domain's upper surface.

Assuming a linearised equation of state (Eq. **??**), the dimensionless form of the conservation of mass equation under the Anelastic Liquid Approximation (ALA) can be expressed as (e.g., **?**):

$$\nabla \cdot (\bar{\rho}\mathbf{u}) = 0, \tag{A7}$$

where $\mathbf{u}$ is the velocity. Neglecting inertial terms, the force balance equation becomes:

$$\nabla \cdot \left[\mu\left(\nabla\mathbf{u} + \nabla\mathbf{u}^T - \frac{2}{3}\nabla\cdot\mathbf{u}\mathbf{I}\right)\right] - \nabla p' - Ra\bar{\rho}\hat{\boldsymbol{k}}\bar{\alpha}T' - \frac{Di}{\gamma_0}\frac{c_{p_0}}{c_{v_0}}\bar{\rho}\hat{\boldsymbol{k}}\bar{\chi}_T p' = 0, \tag{A8}$$

where μ denotes the dynamic viscosity, $\mathbf{I}$ the identity tensor, Ra the Rayleigh number, and Di the dissipation number given by, respectively:

$$Ra = \frac{\rho_0\alpha_0\Delta T g_0 d^3}{\mu_0\kappa_0}; \quad Di = \frac{\alpha_0 g_0 d}{c_{p_0}}, \tag{A9}$$

with κ denoting the thermal diffusivity, d the length scale and ΔT the temperature scale. Note that the ~~final~~ last but one term in Eq. **??** is expressed in terms of the temperature perturbation, T' (sometimes called the potential temperature). Finally, in the absence of internal heating, conservation of energy is expressed as:

$$\bar{\rho}\bar{c}_p\left(\frac{\partial T'}{\partial t} + \boldsymbol{u}\cdot\nabla T'\right) - \nabla\cdot\left[\bar{k}\nabla(\bar{T} + T')\right] + Di\bar{\alpha}\bar{\rho}g\cdot\mathbf{u}T' - \frac{Di}{Ra}\Phi = 0, \tag{A10}$$

where k is the thermal conductivity and Φ denotes viscous dissipation.

Case	Discretisation	Resolution	DOF ($\boldsymbol{u}$)	DOF (p)	DOF (T)	Nu	V_{RMS}
Base Case ($Ra = 1 \times 10^4$)	Q2Q1:Q2	320×320	821762	103041	410881	4.885	42.86
Base Case ($Ra = 1 \times 10^5$)	Q2Q1:Q2	320×320	821762	103041	410881	10.54	193.21
Base Case ($Ra = 1 \times 10^5$)	$Q2P_{1DG}$:Q2	320×320	821762	307200	410881	10.54	193.21
Base Case ($Ra = 1 \times 10^6$)	Q2Q1:Q2	320×320	821762	103041	410881	22.03	833.99
Base Case ($Ra = 1 \times 10^6$)	Q2Q1:Q1	320×320	821762	103041	103041	21.86	834.10
Compressible	Q2Q1:Q2	320×320	821762	103041	410881	7.575	155.09
Viscoplastic	Q2Q1:Q2	320×320	821762	103041	410881	6.617	79.09
3-D Cartesian	Q2Q1:Q2	$60 \times 60 \times 60$	3294225	141398	1098075	3.539	41.00
2-D Cylindrical Shell	Q2Q1:Q2	2048×512	8396800	1050624	4198400	9.541	193.26
3-D Spherical Shell - Isoviscous	Q2Q1:Q2	98304×64	152175366	6389890	50725122	3.506	32.62
3-D Spherical Shell - $\mu(T)$	Q2Q1:Q2	98304×64	152175366	6389890	50725122	2.922	22.99

Table A1. Highest resolution results from benchmark cases analysed herein. DOF = degrees of freedom, for velocity ($\boldsymbol{u}$), pressure (p) and temperature (T); Nu = surface Nusselt number.

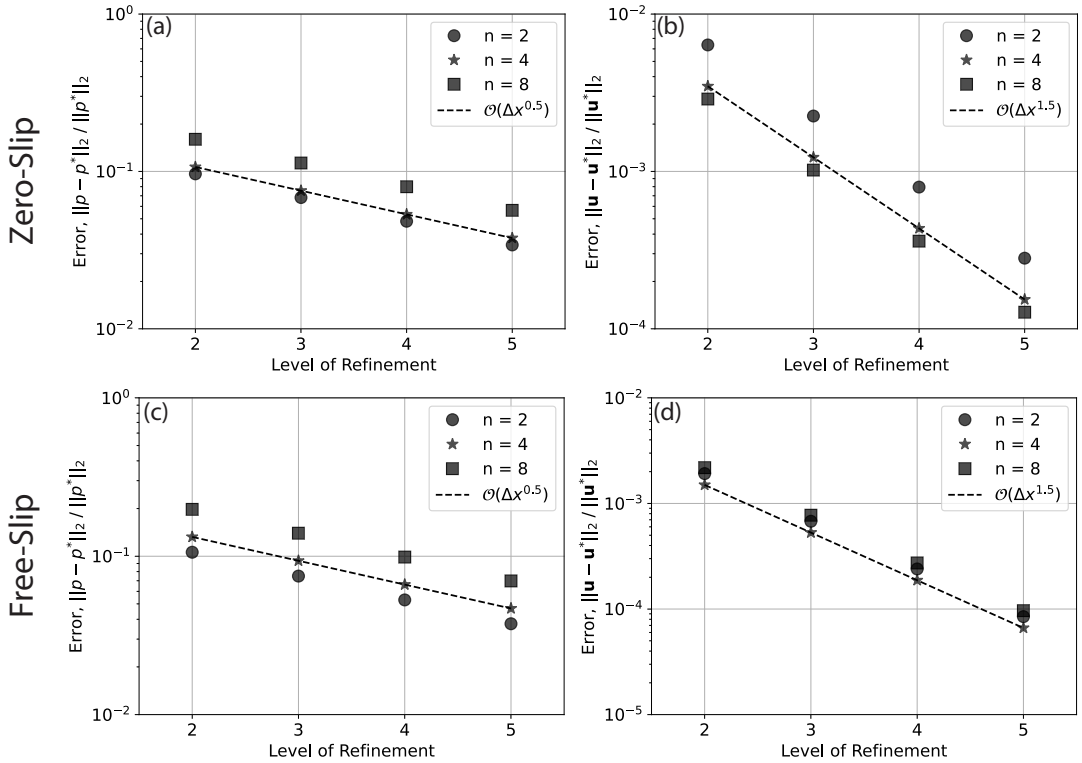

Figure A1. Convergence for 2-D cylindrical shell cases with zero-slip (a-b) and free-slip (c-d) boundary conditions, using a Q2Q1 finite element pair for the Stokes system, driven by a delta-function forcing at different wave-numbers, n, as indicated in the legend (see **?**, for further details). Convergence rate is indicated by dashed lines, with the order of convergence provided in the legend. For the cases plotted, the series of meshes start at refinement level 1, where the mesh consists of 1024 divisions in the tangential direction and 64 radial layers. At each subsequent level the mesh is refined by doubling resolution in both directions.

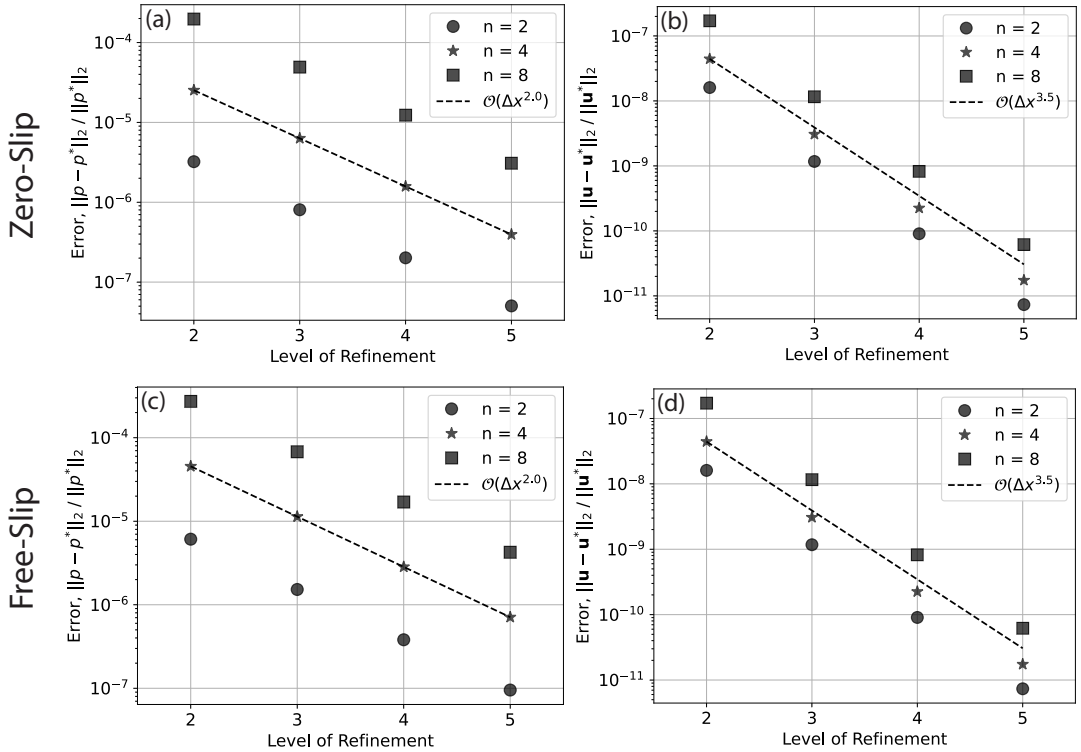

Figure A2. Convergence for 2-D cylindrical shell cases with zero-slip (a-b) and free-slip (c-d) boundary conditions, using a $Q2P_{1DG}$ finite element pair for the Stokes system, driven by a delta-function forcing at different wave-numbers, n, as indicated in the legend (see **?**, for further details). Convergence rate is indicated by dashed lines, with the order of convergence provided in the legend. For the cases plotted, the series of meshes start at refinement level 1, where the mesh consists of 1024 divisions in the tangential direction and 64 radial layers. At each subsequent level the mesh is refined by doubling resolution in both directions.

~~All authors acknowledge support from the Australian Research Data Commons (ARDC: https://ardc.edu.au/, under the G-Adopt platform grant: PL031), AuScope, Geosciences Australia and the National Computational Infrastructure (NCI). DRD and SCK acknowledge support from the Australian Research Council, under grant no. DP170100058. Numerical simulations were undertaken at the NCI National Facility in Canberra, Australia, which is supported by the Australian Commonwealth Government. Authors are grateful to the Firedrake development team, particularly David Ham, for support and advice at various points of this research.~~