Geosci. Model Dev., 6, 2099–2119, 2013
www.geosci-model-dev.net/6/2099/2013/
doi:10.5194/gmd-6-2099-2013

Geoscientific Model Development
Open Access

Automating the solution of PDEs on the sphere and other manifolds in FEniCS 1.2

M. E. Rognes[1], D. A. Ham[2,3], C. J. Cotter[2], and A. T. T. McRae[2,4]

[1]Center for Biomedical Computing, Simula Research Laboratory, P.O. Box 134, 1325 Lysaker, Norway
[2]Department of Mathematics, Imperial College London, London SW7 2AZ, UK
[3]Department of Computing, Imperial College London, London SW7 2AZ, UK
[4]The Grantham Institute for Climate Change, Imperial College London, London SW7 2AZ, UK

Correspondence to: D. A. Ham (david.ham@imperial.ac.uk)

Received: 14 June 2013 – Published in Geosci. Model Dev. Discuss.: 5 July 2013
Revised: 29 October 2013 – Accepted: 31 October 2013 – Published: 17 December 2013

Abstract. Differential equations posed over immersed manifolds are of particular importance in studying geophysical flows; for instance, ocean and atmosphere simulations crucially rely on the capability to solve equations over the sphere. This paper presents the extension of the FEniCS software components to the automated solution of finite element formulations of differential equations defined over general, immersed manifolds. We describe the implementation and, in particular detail, how the required extensions essentially reduce to the extension of the FEniCS form compiler to cover this case. The resulting implementation has all the properties of the FEniCS pipeline and we demonstrate its flexibility by an extensive range of numerical examples covering a number of geophysical benchmark examples and test cases. The results are all in agreement with the expected values. The description here relates to DOLFIN/FEniCS 1.2.

1 Introduction

The computation of approximate numerical solutions to partial differential equations (PDEs) is an integral component of computational science. At the same time, the traditional development of software for the numerical solution of PDEs is time-consuming and error-prone. However, the FEniCS Project (http://fenicsproject.org, Logg et al., 2012b) offers a radical alternative to the traditional development model. Instead of writing low-level model code in a compiled language such as Fortran or C++, the discretisation of the PDE is expressed in a high level language (the Unified Form Language, UFL; Alnæs, 2012) and the corresponding low-level code is generated automatically by a specialised compiler (Logg et al., 2012d). The impact of this approach is dramatic: models which require tens of thousands of lines of C++ or Fortran, and which take months or years to develop can be written in tens to hundreds of lines of high-level code and developed in days to weeks.

The simulation of geophysical fluids has particular features; for instance, the solution of PDEs on the surface of the sphere is of particular significance for the simulation of flow in the ocean and atmosphere. Prior to version 1.2, the FEniCS software has only supported finite element discretisations defined over meshes of the same geometric and topological dimension. As such, the required feature (i.e. the possibility to define discretisations over immersed manifolds such as the sphere) has been missing.

In this paper, we detail the extension of the FEniCS software to enable this feature and as a consequence a multitude of geophysical flow simulation scenarios. We achieve this by extending the FEniCS software components to appropriately handle general two-dimensional manifolds in three-dimensional space, and to general one-dimensional manifolds in two- and three-dimensional spaces. This extension essentially reduces to the ability to evaluate all of the relevant integrals over an element immersed in a higher dimensional space, and to giving the correct definitions to the language elements of UFL in the manifold context. Although the sphere is of particular significance in geoscientific modelling, choosing to solve the more general manifold problem provides additional flexibility and utility. This

generality enables the support of oblate spheroids, as well as a wide range of manifold geometries in other application areas across science and engineering.

A number of other finite element software libraries support solving equations over immersed manifolds, including ALBERTA (Schmidt et al., 2005), DUNE-FEM (including support for div- and curl-conforming finite element spaces) (Dedner et al., 2010), Nektar++ (Sherwin et al., 2013), and deal.II (DeSimone et al., 2009). In addition, high-order discontinuous Galerkin methods have been implemented on manifolds as part of the SLIM ocean model project (Bernard et al., 2008). In contrast to these libraries, however, the FEniCS software heavily relies on and draws its primary advantage from special-purpose finite element code generation. In our description of the implementation here, we therefore focus on the extension of the code generation pipeline to the immersed manifolds case. This implementation aspect extends and differs from that of existing tools, and constitutes a main contribution of this work.

This paper is organised as follows. In Sect. 2, we summarise various aspects of the mathematical formulation of finite element methods over immersed manifolds, including definitions of pullbacks of scalar and vector fields, and differential operators. The key implementation aspects of the required extensions to the FEniCS software are presented in Sect. 3. Section 4 considers verification aspects and Sect. 5 further describes a wide range of numerical examples and tests. We comment on the scope of the current implementation, including limitations and natural extensions, in Sect. 6, before detailing where the implementation and the Supplement can be found in Sects. 7, and 8, and providing some concluding remarks in Sect. 9.

2 Mathematical formulation

This section summarises the distinctive mathematical features of finite element formulations defined over computational domains that are immersed manifolds. The mathematical formulation adopted will be detailed in increasing complexity, beginning with the simplest finite element projection for scalar-valued quantities, and then introducing differential operators and vector-valued functions. The material has deliberately been kept at a minimal level of complexity; for readers more interested in the mathematical theory of manifolds, we recommend for instance Barden and Thomas (2003) or Holm (2008).

Throughout this section, we let Ω be a smooth m-dimensional manifold immersed in $\mathbb{R}^n$, with $m \leq n$. For simplicity, we also let $1 \leq m$ and $n \leq 3$. We will refer to m as the manifold dimension or topological dimension, and to n as the physical or geometric dimension. We approximate this manifold by a piecewise linear tessellation of simplices (intervals in one topological dimension, triangles in two topological dimensions, or tetrahedra in three topological dimensions) $\mathcal{T} = \{T\}$. In particular, each simplex cell T in the mesh $\mathcal{T}$ will then have topological dimension m and geometric dimension n.

2.1 Galerkin projection on the manifold

The finite element method is founded on the concept of finite element spaces. A finite element space V is defined to contain all functions that have some specified polynomial expansion in each cell of the mesh, together with some specified continuity constraint between neighbouring cells. Broadly speaking, the finite element discretisation of a partial differential equation can be described as the projection of that equation onto some finite element space V. The Galerkin projection of a function f onto a finite element space V is a basic finite element operation and defined as the function v in V such that

$$\int_{\mathcal{T}} vw \mathrm{d}x = \int_{\mathcal{T}} fw \mathrm{d}x, \tag{1}$$

for all test functions w in V. If V is N-dimensional with basis $\{\phi_j\}_{j=1}^N$, then we may write

$$v = v_j \phi_j, \tag{2}$$

where $\{v_j\}$ are the expansion coefficients of v relative to the basis $\{\phi_j\}$. Here, and in the rest of the paper, we follow the Einstein summation convention in which summation occurs over an index repeated within a product. Taking $w = \phi_i$ in Eq. (1) for $i = 1, \ldots, N$, we obtain a finite dimensional linear system for the expansion coefficients v_j:

$$M_{ij} v_j = b_i, \tag{3}$$

having defined

$$M_{ij} = \int_{\mathcal{T}} \phi_i \phi_j \mathrm{d}x = \sum_{T \in \mathcal{T}} \int_T \phi_i \phi_j \mathrm{d}x, \tag{4}$$

and

$$b_i = \int_{\mathcal{T}} f \phi_i \mathrm{d}x = \sum_{T \in \mathcal{T}} \int_T f \phi_i \mathrm{d}x. \tag{5}$$

Moreover, for each $T \in \mathcal{T}$, we label the local integral contributions

$$M_{T,ij} = \int_T \phi_i \phi_j \mathrm{d}x, \tag{6}$$

and

$$b_{T,i} = \int_T f \phi_i \mathrm{d}x. \tag{7}$$

In view of Eqs. (4) and (5), the *assembly* of the operators M and b reduce to the evaluation of sums of certain

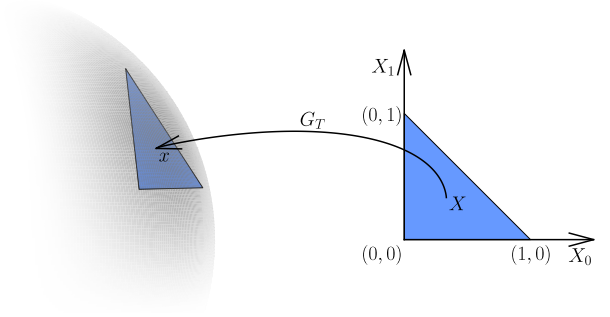

Fig. 1. The transform G_T maps the reference cell T_0 to cell T, which in this case approximates part of a sphere. The point X in reference space is mapped to the point x in physical space: $x = G_T(X)$.

integrals over individual cells $T \in \mathcal{T}$. This procedure is the standard assembly strategy for the finite element method. For more details on finite element assembly, the reader is directed to Logg et al. (2012c) or any standard text on the finite element method (for example Zienkiewicz et al., 2005; Karniadakis and Sherwin, 1999).

2.1.1 Change of coordinates

A change of coordinates to a reference cell T_0 offers a standard and efficient evaluation procedure for each of the local contributions in Eqs. (6) and (7). Recalling that each cell $T \subset \mathbb{R}^n$ is of topological dimension m, we define a fixed reference cell $T_0 \subset \mathbb{R}^m$ and assume that there exists a mapping G_T such that $T = G_T(T_0)$. We write here and throughout $X = (X_1, \ldots, X_m)$ for the coordinates of a point in reference space and $x = (x_1, \ldots, x_n)$ for the coordinates in physical space. Figure 1 illustrates this mapping and the notation employed.

Similarly, we will employ lower case Greek letters for basis functions in physical space, and the corresponding upper case letters for the pullback of those functions to the reference cell. For scalar-valued functions, the pullback is through function composition:

$$\Phi_i(X) = \phi_i(x) = \phi_i\left(G_T(X)\right). \tag{8}$$

Using the definitions above and the usual change of coordinate rules, Eq. (6) becomes

$$\begin{aligned}\int_T \phi_i(x)\phi_j(x)\mathrm{d}x &= \int_T \phi_i(G_T(X))\phi_j(G_T(X))\mathrm{d}x \\ &= \int_{T_0} \Phi_i(X)\Phi_j(X)|\mathbf{J}_T|\mathrm{d}X,\end{aligned} \tag{9}$$

where $\mathbf{J}_T$ is the Jacobian of the transformation G_T and $|\mathbf{J}_T|$ is the Jacobian determinant.

2.1.2 The Jacobian and its pseudo-determinant

The derivation in Eq. (9) applies for both the standard case $m = n$ and the immersed manifold case where $m < n$. The only difference for the latter case is the generalised definitions of the Jacobian and its determinant. In general, the Jacobian $\mathbf{J}$ of the transform $G : \mathbb{R}^m \to \mathbb{R}^n$ is given by the matrix

$$\mathbf{J}_{\gamma\tau} = \frac{\partial G(X)_\gamma}{\partial X_\tau} = \frac{\partial x_\gamma}{\partial X_\tau} \qquad \gamma = 1, \ldots, n, \quad \tau = 1, \ldots, m. \tag{10}$$

Note that τ varies over the manifold dimension m, which is also the geometric and topological dimension of the reference cell, while γ varies over the physical dimension n. To make this concrete, the Jacobian for a two-dimensional manifold immersed in $\mathbb{R}^3$ is given by

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x_1}{\partial X_1} & \frac{\partial x_1}{\partial X_2} \\ \frac{\partial x_2}{\partial X_1} & \frac{\partial x_2}{\partial X_2} \\ \frac{\partial x_3}{\partial X_1} & \frac{\partial x_3}{\partial X_2} \end{bmatrix}. \tag{11}$$

For affine transformations G_T, the Jacobian $\mathbf{J}_T$ will be constant over each cell T. For non-affine transformations, for instance in the case of curved cells, the Jacobian will vary as a function of X.

The Jacobian pseudo-determinant is the transformation of the volume of the differential integral measure. For a one-dimensional manifold, this is the length of the single column vector of $\mathbf{J}$, while, for a two-dimensional manifold, this is the volume of the parallelogram spanned by the two columns of $\mathbf{J}$. More precisely, writing the Jacobian in terms of its column vectors $\mathbf{J} = [\boldsymbol{J}_1, \boldsymbol{J}_2, \ldots, \boldsymbol{J}_m]$, we have

$$|\mathbf{J}| = \begin{cases} |\boldsymbol{J}_1|_2 & m = 1 \\ |\boldsymbol{J}_1 \times \boldsymbol{J}_2|_2 & m = 2, \end{cases} \tag{12}$$

where $|\cdot|_2$ denotes the Euclidean norm. The pseudo-determinant employed here is the square root of the Gram determinant (Kuptsov, 2011). Note that, in the $n = m$ case, this reduces to the absolute value of the usual definition of the determinant.

2.2 Derivatives on the manifold

In order to evaluate more complicated variational forms, it is necessary to be able to evaluate derivatives of functions defined on the manifold. As before, it is sufficient only to consider the case of a basis function defined on a single cell, since all integrals will be decomposed into sums of integrals over basis functions on single cells.

Suppose we have some function $\phi(x)$ defined on an cell $T \subset \mathbb{R}^n$ with pullback $\Phi(X)$ defined on the reference cell $T_0 \subset \mathbb{R}^m$. The gradient of Φ in reference space is immediate:

$$(\nabla_X \Phi(X))_\tau = \frac{\partial \Phi(X)}{\partial X_\tau} \qquad \tau = 1, \ldots, m. \tag{13}$$

Define the tangent space of cell T as the image of the corresponding Jacobian $\mathbf{J}$ over reference space; thus, any v in the tangent space can be written as $v = \mathbf{J}V$ for some V in reference space. We define the gradient of ϕ in physical space $\nabla_x \phi$ via the usual Gâteaux directional derivative:

$$\nabla_x \phi(x) \cdot v = \lim_{\epsilon \to 0} \frac{\phi(x + \epsilon v) - \phi(x)}{\epsilon} \tag{14}$$

for any v in the tangent space.

Assume that the mapping G is affine and non-degenerate, such that the columns of $\mathbf{J}$ are linearly independent. It follows from the definitions above that

$$\begin{aligned} \nabla_X \Phi(X) \cdot V &= \lim_{\epsilon \to 0} \frac{\Phi(X + \epsilon V) - \Phi(X)}{\epsilon} \\ &= \lim_{\epsilon \to 0} \frac{\phi(x + \epsilon v) - \phi(x)}{\epsilon} = \nabla_x \phi(x) \cdot v. \end{aligned} \tag{15}$$

Next, let $\mathbf{J}^\dagger$ denote the Moore–Penrose pseudo-inverse of $\mathbf{J}$ (Penrose, 1955), given in this case by

$$\mathbf{J}^\dagger = \left(\mathbf{J}^{\mathrm{T}}\mathbf{J}\right)^{-1}\mathbf{J}^{\mathrm{T}}, \tag{16}$$

where the superscript T denotes the transpose. Then clearly, for $v = \mathbf{J}V$,

$$\mathbf{J}^\dagger v = \left(\mathbf{J}^{\mathrm{T}}\mathbf{J}\right)^{-1}\left(\mathbf{J}^{\mathrm{T}}\mathbf{J}\right)V = V. \tag{17}$$

Inserting Eq. (17) into Eq. (15), and rearranging, we find that

$$\nabla_x \phi(x) \cdot v = (\mathbf{J}^\dagger)^{\mathrm{T}} \nabla_X \Phi(X) \cdot v. \tag{18}$$

In our implementation, vector quantities are always represented as elements of the n-dimensional space in which the manifold is immersed. In this representation, we additionally require that

$$\nabla_x \phi(x) \cdot \boldsymbol{k} = 0, \tag{19}$$

where $\boldsymbol{k}$ is the unit normal vector to the cell T, and hence we obtain the n-dimensional vector

$$\nabla_x \phi(x) = (\mathbf{J}^\dagger)^{\mathrm{T}} \nabla_X \Phi(X). \tag{20}$$

From Eq. (16), it follows immediately that the column space of $(\mathbf{J}^\dagger)^{\mathrm{T}}$ coincides with that of $\mathbf{J}$. We therefore observe that $\nabla_x \phi(x)$ is in the tangent space of cell T as expected. In the special case of a one-dimensional manifold ($m = 1$), the pseudo-inverse reduces to

$$\mathbf{J}^\dagger = \frac{\mathbf{J}^{\mathrm{T}}}{|\mathbf{J}|^2}. \tag{21}$$

2.2.1 The weak Laplacian

To illustrate the practical implications of the above, we examine the integral form corresponding to the weak Laplacian over a pair of basis functions ϕ_i and ϕ_j on a single cell T:

$$\int_T \nabla \phi_i \cdot \nabla \phi_j \mathrm{d}x. \tag{22}$$

Applying Eq. (20) and the change of integration measure, we immediately find that

$$\begin{aligned} &\int_T \nabla_x \phi_i(x) \cdot \nabla_x \phi_j(x) \mathrm{d}x = \\ &\quad \int_{T_0} \left((\mathbf{J}^\dagger)^{\mathrm{T}} \nabla_X \Phi_i(X)\right) \cdot \left((\mathbf{J}^\dagger)^{\mathrm{T}} \nabla_X \Phi_j(X)\right) |\mathbf{J}| \mathrm{d}X. \end{aligned} \tag{23}$$

So, as before, the integrals over cells in the mesh may be evaluated on the reference cell using the Jacobian and, in this case, its pseudo-inverse. Observe that $(\mathbf{J}^\dagger)^{\mathrm{T}} \nabla_X \Phi_j(X)$ has dimension n, and that the index in the inner product in Eq. (23) therefore runs from 1 to n.

2.3 Constructing vector-valued fields on the manifold

Recall that, in our implementation, vector quantities are always represented as elements of the n-dimensional space in which the manifold is immersed. In this representation, there are two distinct forms of finite element space for vector-valued quantities employed in the finite element method. In the most simple case, the finite element space is the Cartesian product of scalar-valued spaces: each component of the vector varies independently as a piecewise polynomial over each cell. The finite element space may be continuous at cell boundaries, in which case all components will be continuous, or discontinuous, in which case no continuity is enforced at cell boundaries for any component of the vector value. As a matter of notation, we will write CG_k^n for the space of continuous n-dimensional vector fields with polynomial degree k, and DG_k^n for the corresponding space with no inter-element continuity constraint.

Recall that $\mathcal{T} = \{T\}$ is the tessellation of the m-dimensional manifold in $\mathbb{R}^n$ by m-simplices. A vector field represented in this way has n components, where n is the dimension of the space in which the manifold is immersed. For instance, a vector field of this type on a tessellation of the surface of the sphere will have three components, not two. This has the natural consequence that the vector field is not constrained to be tangent to the manifold. Where this is required, it will have to be imposed as an additional constraint in the equations to be solved. There is an example illustrating this in Sect. 5.1.2.

Since these finite element spaces are the Cartesian products of scalar-valued component spaces, the basis functions for the vector field can be written with respect to the scalar basis functions. For example, if $\{\phi_j\}_{j=1}^N$ is the basis for a scalar-valued space, the basis for the corresponding two-dimensional vector space is given by

$$\{\boldsymbol{\phi}_i\}_{i=1}^{2N} = \left\{\begin{bmatrix}\phi_j \\ 0\end{bmatrix}\right\}_{j=1}^{N} \bigcup \left\{\begin{bmatrix}0 \\ \phi_k\end{bmatrix}\right\}_{k=1}^{N}. \tag{24}$$

The pullback through the map from the reference cell G_T is applied separately to each Cartesian component:

$$\boldsymbol{\Phi}_i(X) = \boldsymbol{\phi}_i(x) = \boldsymbol{\phi}_i(G_T(X)). \tag{25}$$

Consequently, the mass integral over a single cell transforms in the same manner as the scalar case:

$$\int_T \boldsymbol{\phi}_i(x) \cdot \boldsymbol{\phi}_j(x) \mathrm{d}x = \int_{T_0} \boldsymbol{\Phi}_i(X) \cdot \boldsymbol{\Phi}_j(X) |\mathbf{J}_T| \mathrm{d}X. \tag{26}$$

The Cartesian product vector spaces are, in fact, a special case of a more general class of mixed finite element spaces which can be composed of any other finite element spaces. If U and V are finite element spaces of any type with bases $\{\phi_j\}_{j=1}^N$ and $\{\psi_k\}_{k=1}^M$, then $W = U \times V$ is the Cartesian product of these spaces with basis given by

$$\{\omega_i\}_{i=1}^{N+M} = \left\{\begin{bmatrix}\phi_j \\ 0\end{bmatrix}\right\}_{j=1}^{N} \bigcup \left\{\begin{bmatrix}0 \\ \psi_k\end{bmatrix}\right\}_{k=1}^{M}. \tag{27}$$

This definition is fully recursive so any number of spaces of any type can be combined in this way. Mixed spaces require no special handling in the manifold case beyond that required by the component spaces. That is to say, without loss of generality, if ω is a basis function of W of the form $\omega = \begin{bmatrix}\phi & 0\end{bmatrix}^T$ then its pullback Ω is given by $\Omega = \begin{bmatrix}\Phi & 0\end{bmatrix}^T$, where Φ is the pullback of ϕ.

Vector-valued finite element spaces can, as we have just seen, be constructed via Cartesian products of scalar finite element spaces. However, there are also a collection of highly useful finite element families that are inherently vector-valued. In the geoscientific context, the most common example of such is the lowest order Raviart–Thomas element (Raviart and Thomas, 1977), known to the finite volume community as the C-grid velocity discretisation (Arakawa and Lamb, 1977). Other examples include the Nédélec edge and face elements (Nédélec, 1980, 1986). In both cases we compute in Cartesian coordinates, with the metric terms being formed implicitly through the transformation from the reference element.

We have already defined the grad operator on a manifold. The vector calculus operators div and curl on a two-dimensional manifold M are most easily defined as limits of flux and circulation integrals,

$$\operatorname{div} \boldsymbol{u}(x) = \lim_{\epsilon \to 0} \frac{1}{|C_\epsilon|} \oint_{C_\epsilon} \boldsymbol{u} \cdot \boldsymbol{n} \mathrm{d}x, \tag{28}$$

$$\operatorname{curl} \boldsymbol{u}(x) = \lim_{\epsilon \to 0} \frac{1}{|C_\epsilon|} \oint_{C_\epsilon} \boldsymbol{u} \cdot \mathrm{d}x, \tag{29}$$

where C_ϵ is a loop centred on x that approaches a circle of radius ϵ as $\epsilon \to 0$, and $|C_\epsilon|$ is the area on the manifold enclosed by C_ϵ.

Vector fields $\boldsymbol{u}$ from divergence-conforming (*div-conforming*) finite element spaces (such as the Raviart–Thomas finite element space) are constrained so that the normal component $\boldsymbol{u} \cdot \boldsymbol{n}$ is continuous across each facet of the tessellation, where $\boldsymbol{n}$ is the normal vector on the facet. The tangential component(s) are not required to be continuous. There is sufficient continuity for the divergence operator to be globally defined, hence the term "div-conforming". For such element spaces, vector fields are naturally mapped from a reference cell to each physical cell via the *contravariant Piola* transform (Brezzi and Fortin, 1991; Rognes et al., 2009): letting $\boldsymbol{\Phi}$ be a vector field defined on the reference cell T_0, we define the field $\boldsymbol{\phi}$ on the physical cell T as

$$\boldsymbol{\phi}(x) = \frac{1}{\pm |\mathbf{J}_T|} \mathbf{J}_T \, \boldsymbol{\Phi}(X). \tag{30}$$

We remark that, in the case of a m-dimensional manifold immersed in $\mathbb{R}^n$, $\boldsymbol{\Phi}$ is a vector field with m components and Eq. (30) defines $\boldsymbol{\phi}$ as an n-vector field. Moreover, observe that $\boldsymbol{\phi}$ is in the tangent space of T by construction. The sign of Eq. (30) is positive if the current element has the same orientation as the manifold and negative if the orientations differ. On a non-orientable manifold, the sign is indeterminant and the contravariant Piola transformation cannot be employed. The implementation of manifold orientation is discussed in Sect. 3.3.2.

Conversely, *curl-conforming* finite element spaces, such as Nédélec spaces, are defined such that for each field in this space the component of the field tangent to each facet is continuous across that facet, while the normal component to the facet may be discontinuous. The desired tangential continuity is enabled if the fields are mapped from a reference cell to each physical cell via the *covariant Piola* transform:

$$\boldsymbol{\phi}(x) = (\mathbf{J}_T^\dagger)^{\mathrm{T}} \, \boldsymbol{\Phi}(X). \tag{31}$$

We note that the covariant Piola transform also maps m-vector fields to n-vector fields, and that its image is in the tangent space of T, by definition, since the column space of $(\mathbf{J}_T^\dagger)^{\mathrm{T}}$ coincides with the column space of $\mathbf{J}_T$ as previously noted in Sect. 2.2.

A third approach to obtaining vector fields on manifolds is the approach to high-order discontinuous Galerkin methods on manifolds in Bernard et al. (2008), under which vector fields are expanded using a local tangent basis on each

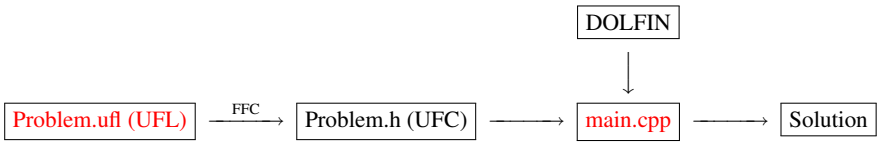

Fig. 2. The FEniCS pipeline viewed from the C++ interface. In the Python interface, the components UFL, FFC, UFC and DOLFIN are more closely integrated.

element, but the surface fluxes are resolved into three dimensions. We did not implement this approach since it would require more invasive changes in UFC.

2.4 Facet integrals

Suppose $u = u_i \psi_i$ is a scalar field and $\boldsymbol{v} = v_j \boldsymbol{\phi}_j$ is a vector field. A commonly occurring integral form (for example for the pressure gradient in a mixed finite element fluid simulation) is

$$\int_{\mathcal{T}} \nabla u \cdot \boldsymbol{v} \mathrm{d}x. \tag{32}$$

A routine manipulation, for example to impose boundary conditions or to introduce coupling between elements if the spaces are discontinuous, is to integrate by parts:

$$\int_{\mathcal{T}} \nabla u \cdot \boldsymbol{v} \mathrm{d}x = -\int_{\mathcal{T}} u \nabla \cdot \boldsymbol{v} \mathrm{d}x + \int_{\Gamma} u \boldsymbol{v} \cdot \boldsymbol{n} \mathrm{d}s + \int_{\Gamma^0} u^+ \boldsymbol{v}^+ \cdot \boldsymbol{n}^+ + u^- \boldsymbol{v}^- \cdot \boldsymbol{n}^- \mathrm{d}s \,. \tag{33}$$

Here, Γ indicates the surface of $\mathcal{T}$ (empty if the manifold is closed), and Γ^0 is the set of interior facets (points in one dimension, edges in two dimensions and faces in three dimensions) between cells in $\mathcal{T}$. $\boldsymbol{n}$ is the outward pointing normal to the element in question, with the superscripts $+$ and $-$ denoting the two sides of each interior facet. In the manifold case, there are two features of facet normals which are significant. The first is that, as with other vector-valued quantities, the facet normal has dimension n, that of the physical space. The second is that adjacent cells on a manifold are not typically coplanar ($m = 2$) or colinear ($m = 1$). Consequently, the identity $\boldsymbol{n}^+ = -\boldsymbol{n}^-$, which holds in the standard $n = m$ case, does not generally hold on a manifold.

3 Integrating manifolds into the FEniCS Project pipeline

3.1 The FEniCS Project pipeline

The FEniCS Project is a collection of numerical software, supported by a set of novel algorithms and techniques, aimed at the automated solution of differential equations using finite element methods (Logg et al., 2012b). The FEniCS Project software consists of a number of interoperable software components which define a full computational pipeline when used together.

The core of the FEniCS pipeline is the following (cf. Fig. 2). Consider the common use case where a finite element formulation of a partial differential equation is given in mathematical form and the numerical solution is the desired output. The simplest such example is the Galerkin projection problem Eq. (1) over, for instance, the space of piecewise linear functions, defined relative to the tessellation $\mathcal{T}$, to obtain the discrete solution u.

The first step is to express the variational formulation in the domain-specific Python-embedded language Unified Form Language (UFL) (Alnæs et al., 2013; Alnæs, 2012). Continuing with Eq. (1) as an example, to express the variational formulation the user must express (i) the finite element space V; (ii) the basis functions u, v and coefficient f involved; and (iii) the right-hand and left-hand side variational forms.

The next step in the pipeline is the processing of the UFL specification by a special purpose compiler: the FEniCS Form Compiler (FFC) (Kirby and Logg, 2006; Logg et al., 2012d). FFC is targeted at generating efficient, lower-level code for the assembly of the relevant finite element tensors. FFC generates code that conforms to the Unified Form-assembly Code (UFC) interface (Alnæs et al., 2009, 2012). More precisely, given a set of variational forms specified in UFL, separate classes are generated for each of the finite elements over which the basis functions and any coefficients are defined, for each of the variational forms, and for each of the integrals appearing in the forms. The finite element classes then provide functions for evaluating the specific basis functions, computing the specific local-to-global degree of freedom maps, evaluating the specific degrees of freedom on arbitrary functions, et cetera. The integral class(es) similarly provide functions for computing the specific local element tensor.

The generated code can then be used by the user directly, or, as is more common, be used via the problem-solving environment and finite element library DOLFIN. DOLFIN provides high performance computing functionality for simplicial meshes, automated assembly of variational forms, relying on the generated code for each specific form as

```
# Define triangle cell embedded in R^3
cell = Cell("triangle", 3)

# Define Lagrange element over this cell
Q = FiniteElement("Lagrange", cell, 1)
# Define Lagrange vector element
V = VectorElement("Lagrange", cell, 1)

# Arguments defined over V will have 3 components:
u = Coefficient(V)
u[0], u[1], u[2]
```

Fig. 3. UFL code defining scalar and vector finite elements on a triangle embedded in $\mathbb{R}^3$. This code is included in the Supplement as snippets/finiteelement.ufl.

detailed above, and interfaces to numerical linear algebra libraries; for more details see Logg and Wells (2010) and Logg et al. (2012e). DOLFIN provides both a C++ interface and a Python interface. In the Python interface, the steps detailed above are all closely integrated; in particular, the code generation happens seamlessly via a just-in-time (JIT) compilation process in which C++ code is generated, compiled and executed at runtime within a single Python function call. This process is described in more detail in Logg et al. (2012d).

Three implementation points crystallise as necessary in order to integrate weak formulations defined over manifolds into the FEniCS pipeline.

- Extend the form language UFL to include finite elements and variational forms defined over simplicial cells of differing geometric and topological dimension.
- Extend the form compiler FFC to provide corresponding support for the novel UFL finite elements and forms introduced.
- Support UFC and DOLFIN data structures for, and operations over, meshes defined over simplicial cells with differing geometric and topological dimension.

The extensions to the previous functionality required to achieve these three aspects are described below.

3.2 Extending and interpreting UFL over manifolds

As described in Alnæs et al. (2013), UFL is in essence composed of three sublanguages for expressing (i) finite elements; (ii) expressions, including terminal types and operators acting on them; and (iii) variational forms.

3.2.1 Finite and mixed finite elements over manifolds

A basic UFL finite element is defined in terms of a family, a cell and a (polynomial) degree. In order to allow finite elements to be defined over manifolds, the only modification required is to define a cell of differing geometric and topological dimensions. Such cells are in place in UFL for $m \leq n = 1, 2, 3$. Geometric quantities, such as the volume, the circumradius or facet normals, are associated with each cell type. When appropriate, these are defined relative to the topological dimension of the cell; for instance, the volume of a triangle cell embedded in $\mathbb{R}^3$ refers to the two-dimensional volume of the cell.

UFL allows Cartesian combinations (and nested combinations) of finite elements of arbitrary families and degrees to form vector, tensor or mixed elements with an arbitrary number of components. The number of components (the value dimension) of a vector element defaults to the geometric dimension n of the cell over which the element is defined. Similarly the shape of a tensor element defaults to (n, n). The UFL code listing in Fig. 3 illustrates this. Note that UFL mixed elements (including vector and tensor elements) are defined over a common cell. As a consequence, mixed elements for which different components are defined over different cells are not supported.

We emphasise that UFL vector elements are generally not constrained to lie in the tangent space of the manifold. This is a deliberate choice for the sake of flexibility, applicability and consistency. For applications where the vector fields should be restricted to the tangent space, this requirement can be enforced either via an additional variational constraint, or, if applicable, by employing div-conforming or curl-conforming finite elements. Note however that the basis functions and coefficients defined over the latter are still indexed from $0, \ldots, n-1$ where n is the geometric dimension of the cell and value dimension of the element.

3.2.2 Differential operators over manifolds in UFL

A variational form is typically defined, both mathematically and in UFL, via a set of operators acting on a set of basis functions or coefficients integrated over some domain. Taking Eq. (22) as an example, the differential operator ∇_x acts on basis functions ϕ_i and ϕ_j. The operators defined by UFL

```
cell = Cell("triangle", 3)
V = FiniteElement("Lagrange", cell, 1)
u = TrialFunction(V)
v = TestFunction(V)

a = inner(grad(u), grad(v))*dx
# or equivalently
# a = sum(u.dx(i)*v.dx(i) for i in range(3))
```

Fig. 4. UFL code defining the weak Laplacian operator from Eq. (22) for piecewise linear elements over a triangle embedded in $\mathbb{R}^3$. This code is included in the Supplement as snippets/laplacian.ufl.

include arithmetic, algebraic, indexing and differential operators. The arithmetic and algebraic operators extend trivially to the case of functions defined over manifolds; on the other hand, the precise extensions of the differential operators deserve a few comments.

UFL provides the differential operators `grad`, `div`, `curl`, and `rot`. In addition, component-wise derivatives can be expressed via `dx(i)` or `Dx(u, i)` for some function `u` and index `i`, which ranges over the n Cartesian components of the gradient vector represented in the embedded space. The UFL gradient `grad` can be viewed as the base operator: it is defined in accordance with Eq. (14), which, in particular, defines ∇u as an element of $\mathbb{R}^n$. As such, for a scalar-valued basis function `u` defined over a cell of geometric dimension n, `grad(u)` is a vector-valued expression, indexable by an index `i` ranging from 0 to n. Moreover, we define

$$\texttt{grad(u)[i]} := \nabla(u)_i, \tag{34}$$

where ∇u is represented in the n-dimensional physical space; the extension to gradients of vector- and tensor-valued expressions, expanded in n-dimensional Cartesian components, is immediate. Building on Eq. (34), we define the component derivatives `dx(i)` and `Dx(u, i)` as

$$\texttt{u.dx(i)} \equiv \texttt{Dx(u, i)} := \texttt{grad(u)[i]}. \tag{35}$$

In short, the component derivatives are defined as the components of the gradient, and components are defined in terms of the standard Euclidean orthonormal basis for $\mathbb{R}^n$. Figure 4 shows the UFL code corresponding to example Eq. (22).

On an affine triangle, the definition of the divergence operator Eq. (28) simplifies to $\sum_{i=0}^{2}(\nabla u)_i$, since there are no curvature terms. Therefore, in our implementation, we simply define

$$\texttt{div(u)} := \sum_{i=0}^{n-1} \texttt{u.dx(i)}. \tag{36}$$

This would need to be modified if non-affine cells were introduced. The UFL operators `curl` and `rot` (which return identical output in UFL) have not been modified from their definitions for the case $m = n$ and should only be used with care for the case $m < n$. This is because curl is defined using a normal vector, and sometimes the user may wish to use the actual (discontinuous) normal to the mesh manifold, and sometimes they may wish to use a continuous approximation the the normal. An example using a curl operator built using the mesh normal field is given in Sect. 5.5, where it is crucial that this operator produces exactly divergence-free vector fields.

3.2.3 Integration measures over manifolds

UFL supports variational forms defined via integration of an integrand I over a set of predefined classes of domains defined relative to a tessellation $\mathcal{T} = \{T\}$ and sums of such integrals. The more commonly used domains are all cells (`dx`), all exterior facets (`ds`) and all interior facets (`dS`). More precisely, by definition,

$$\texttt{I*dx} := \sum_{T\in\mathcal{T}} \int_T I\,\mathrm{d}x, \quad \texttt{I*ds} := \sum_{e\in\mathcal{E}^e} \int_e I\,\mathrm{d}s,$$
$$\texttt{I*dS} := \sum_{e\in\mathcal{E}^i} \int_e I\,\mathrm{d}s. \tag{37}$$

Here, $\mathcal{E}^e$ refers to the set of all exterior facets of the tessellation $\mathcal{T}$ while $\mathcal{E}^i$ refers to the set of all interior facets. Recall that $\mathcal{T}$ is composed of cells of geometric dimension n and topological dimension m, $1 \le m \le n \le 3$. The facets $\mathcal{T}$, therefore, have geometric dimension n and topological dimension $m-1$, and $\mathrm{d}x$ and $\mathrm{d}s$ in Eq. (37) refer to the standard Lebesgue integration measures on $\mathbb{R}^m$ and $\mathbb{R}^{m-1}$, respectively.

For example, this implies that the integral over all cells of a mesh of the surface of a ball will equal the integral over all exterior facets of a mesh of the ball. Figure 5 illustrates this using DOLFIN code [1].

[1] UFL is not concerned with actual meshes so (Python) DOLFIN code, in which the variational form specification is integrated with the problem solving environment, is used to illustrate here. The essence is the definitions of the forms `a` and `b`.

```python
from dolfin import *

# Define a mesh of a sphere (ball) with radius 1 and a mesh of its
# surface
mesh = Mesh(Sphere(Point(0.0, 0.0, 0.0), 1.0), 8)
surface = BoundaryMesh(mesh, "exterior")

# Integrate 1 over the exterior facets of the mesh of the ball
I = Constant(1.0)
a = I*ds
A = assemble(a, mesh=mesh)

# Integrate 1 over the cells of mesh of the surface of the ball
b = I*dx
B = assemble(b, mesh=surface)

# Confirm that A == B to within numerical precision
eps = 1.e-14
assert (abs(A - B) < eps)
```

Fig. 5. DOLFIN Python code illustrating that an integral over the surface facets of a meshed ball is equivalent to the integral over the manifold mesh of the ball's surface. This example is included in the Supplement as snippets/measures.py.

3.3 Extending the FEniCS Form Compiler (FFC) onto manifolds

The interface of the form compiler FFC has two main entry points: one for compiling a (set of) UFL form(s) and one for compiling a separate UFL finite element.

3.3.1 Compiling variational forms and integrals over manifolds

A UFL form is a sum of UFL integrals each of predefined type determined by the measure symbol. The role of FFC is to generate UFC-compliant code for the form and for each of the integrals. The main part of the integral code is the computation of the local element tensor over a given physical mesh entity for the specific integral. This functionality is provided by the generated code body of the `ufc::*_integral::tabulate_tensor` functions. The UFC specification allows mesh entities and in particular physical cells with differing topological and geometric dimensions. The extension of FFC to immersed manifolds is therefore restricted to extending the generation of the local element tensor code body to this case.

For all integral types, the generated code computes the local element tensors by pulling the integral back to a suitable reference cell as shown in Eq. (9) for the local mass matrix and in Eq. (23) for the local stiffness matrix. For an integral over a cell of topological dimension m, the integral is pulled back to the reference element[2] of both topological and geometric dimension m. FFC uses FIAT (Kirby, 2004) to preevaluate the reference basis functions on the reference cell. Since FIAT only operates on the reference cell, it requires no modification for our purposes.

As demonstrated in Sect. 2, the mathematical representation of the variational forms supported by FFC differs between the standard and immerse manifolds cases only in the definition of the Jacobian and its pseudo-determinant and inverse. FFC's internal representation follows the mathematics, with the consequence that only the final, code generation, stage requires modification for the immersed manifold case.

The code generation stage of FFC relies on predefined code snippets or kernels for computing the Jacobians, the Jacobian (pseudo-)determinants and the Jacobian (pseudo-)inverses. Compiling forms over cells with differing topological and geometric dimension is therefore simply enabled by providing implementations based on the definition Eqs. (12) and (16) for the (remaining) cases $1 \leq m < n \leq 3$. This light-touch modification is compatible with the full range of optimisations which FFC can insert in the generated form code. We show an example of the generated code corresponding to Eq. (23) in Fig. 6. The geometrically defined UFL operands such as facet normal are likewise extended to immersed manifolds by corresponding modified code snippets.

[2]The definition of the UFC reference cells for dimensions 1, 2, 3 are given in Alnæs et al. (2012).

```
/// Tabulate the tensor for the contribution from a local cell
virtual void tabulate_tensor(double*  A,
                             const double * const *  w,
                             const double*  vertex_coordinates,
                             int cell_orientation) const
```

```
  // Compute Jacobian
  double J[6];
  compute_jacobian_triangle_3d(J, vertex_coordinates);

  // Compute Jacobian inverse and determinant
  double K[6];
  double detJ;
  compute_jacobian_inverse_triangle_3d(K, detJ, J);

  // Set scale factor
  const double det = std::abs(detJ);

  // Compute geometry tensor
  const double G0_0_0 = det*(K[0]*K[0] + K[1]*K[1] + K[2]*K[2]);
  const double G0_0_1 = det*(K[0]*K[3] + K[1]*K[4] + K[2]*K[5]);
  const double G0_1_0 = det*(K[3]*K[0] + K[4]*K[1] + K[5]*K[2]);
  const double G0_1_1 = det*(K[3]*K[3] + K[4]*K[4] + K[5]*K[5]);

  // Compute element tensor
  A[0] = 0.5*G0_0_0 + 0.5*G0_0_1 + 0.5*G0_1_0 + 0.5*G0_1_1;
  A[1] = -0.5*G0_0_0 - 0.5*G0_1_0;
  A[2] = -0.5*G0_0_1 - 0.5*G0_1_1;
  A[3] = -0.5*G0_0_0 - 0.5*G0_0_1;
  A[4] = 0.5*G0_0_0;
  A[5] = 0.5*G0_0_1;
  A[6] = -0.5*G0_1_0 - 0.5*G0_1_1;
  A[7] = 0.5*G0_1_0;
  A[8] = 0.5*G0_1_1;
```

Fig. 6. Generated code for the bilinear form corresponding to the weak Laplacian. This code is extracted from snippets/laplacian.h in the Supplement, which was obtained by running FFC on the code in Fig. 4. The array A represents the 3×3 local element tensor for the discrete Laplacian. For a full explanation of FFC and the code it generates, the reader is directed to Logg et al. (2012d).

3.3.2 Compiling finite elements and dofmaps over manifolds

In addition to the code for integrals and forms, FFC generates UFC finite element and dofmap (degree-of-freedom map) classes for all finite elements encountered in a form and when compiling a separate finite element. The primary non-trivial functionality provided by these generated classes are evaluating degrees of freedom on arbitrary coefficients, evaluating basis functions at arbitrary points, and interpolating vertex values. As for forms and integrals, the FFC code generation strategy for these operations relies on pulling the finite element basis functions back to the appropriate reference cell or, in some cases, pushing the degrees of freedom forward from the reference to the physical cell.

In the cases of scalar basis functions Eq. (8) or Cartesian products of these Eq. (25), the implementation of the pull-backs and push-forwards extend trivially from the case with matching topological and geometric dimension to that of differing topological and geometric dimension. For div- and curl-conforming elements, the implementation of the pull-back is based on the Piola contravariant and covariant transforms Eqs. (30) and (31). As for forms, the intermediate compiler representation preserves the semi-symbolic representation given by the mathematical expressions, and so the generated code again simply calls out to the redefined Jacobians, pseudo-determinants and pseudo-inverses.

However, the finite element families mapped via the contravariant Piola transform present one additional novel aspect: the choice of sign in Eq. (30). As detailed in Rognes et al. (2009) for the case of matching topological and geomet-

ric dimensions, using the signed determinant of the Jacobian in combination with the UFC numbering of mesh entities result in the desired normal component continuity. However, as the generalised Jacobian determinant does not carry a sign, the choice of sign in Eq. (30) must be determined by alternative means. In fact, it is not possible to determine the sign based on the geometry of the physical cell only. Therefore, the code generation assumes that the choice of sign is determined via an additional input argument to the relevant UFC functions.

3.4 Extending DOLFIN onto manifolds

DOLFIN natively supports meshes defined over cells with differing topological and geometric dimension. Therefore, the extensions to the form compiler FFC detailed above essentially yield a fully functional FEniCS pipeline for these meshes.

In view of the previous remarks regarding the contravariant Piola finite element families, the DOLFIN meshes have been extended to keep track of cell orientations defined relative to a global normal direction. More precisely, for a triangle cell with vertex coordinates v_0, v_1, v_2 embedded in $\mathbb{R}^3$, we identify its first two edges by $\boldsymbol{e}_0 = v_1 - v_0$ and $\boldsymbol{e}_1 = v_2 - v_0$. The local cell normal $\boldsymbol{n}_l$ is defined by $\boldsymbol{n}_l = \boldsymbol{e}_0 \times \boldsymbol{e}_1$. Given a global (potentially spatially) varying normal field $\boldsymbol{n}$, the cell is identified as "up"-oriented if $\boldsymbol{n}_l$ and $\boldsymbol{n}$ are aligned; that is, if $\boldsymbol{n}_l \times \boldsymbol{n}(v_m) > 0$ for $\boldsymbol{n}$ evaluated at the cell barycentre v_m and "down"-oriented if $\boldsymbol{n}_l \times \boldsymbol{n}(v_m) < 0$. This orientation information is then propagated through the UFC interface to the code generated by FFC. As an ultimate consequence, the contravariant Piola element families are not available for non-orientable manifolds.

3.5 Execution in parallel

One of the key advantages of the FEniCS pipeline is the separation it achieves between different aspects of the software and algorithms of the finite element method. In particular, the implementation of parallel execution is completely separated from the specification of the numerical method. DOLFIN supports MPI and OpenMP parallelism, including hybrids of the two (Richardson and Wells, 2013). The extension of FEniCS to support immersed manifolds therefore required no modifications to the existing parallel support, and conversely, simulations over manifolds are naturally supported in parallel.

4 Verification

The FEniCS components, including the extension to immersed manifolds presented here, are primarily tested and verified via the following means: firstly, automated unit, system, and regression tests; and secondly, numerical experiments testing the observed convergence rates or other numerical properties of certain test cases against theoretically established values. We make some comments on verification via the first class of tests here. To illustrate the use of the implementation and to provide some further evidence towards its correctness, we provide some examples of the second class of tests in Sect. 5 below.

The FFC implementation is tested through a series of sample forms and elements aiming at covering the scope of forms and elements supported by FFC. The compilation of forms and elements are tested in multiple ways, primarily via regression testing. In particular, the generated code is directly compared to previously established references, it is verified that the generated code compiles and runs, and finally the output of the generated code when run with a set of sample input is compared with previously established references. These tests are carried out for the different representations available in FFC, with and without optimisations. Moreover, since the different representations must provide equal results, the results from running the generated code are compared across representations. This procedure thoroughly tests the sample forms and elements included in the test suite. However, we remark that the quality of this verification strongly depends on the coverage of the test suite.

In DOLFIN, in addition to regression tests as for FFC above, unit tests are emphasised: separate functionality is tested on cases of reduced complexity for which the computed answer can be compared to a known, exact value. In particular, for the extensions of the FEniCS functionality described in this manuscript, unit tests were added ensuring that cell orientations are correctly computed and that all geometry computations relating to cell volumes, circumradius, facet normals and facet area, for all combinations of geometric and topological dimensions $1 \leq m \leq n \leq 3$, are correct. A series of tests verifying the correctness of the results from the automated assembly of simple forms defined over finite elements over cells of varying geometric and topological dimension were also included. A simple, but useful, example is comparing the result of $\int_K 1 \mathrm{d}x$ to the area of K for varying domains K of known area.

For the manifolds case, a much used technique for corroborating the correctness of the extended implementation is comparing the case of an immersed smooth surface with vanishing curvature to the equivalent standard, and hence thoroughly tested, flat case. For example, a known test case on the unit square can be repeated for the unit square immersed in $\mathbb{R}^3$ and the result verified. Further tests may be constructed by subjecting the unit square mesh to rigid body transformations in $\mathbb{R}^3$ and solving the problem on the transformed mesh.

All of these tests are available as a part of the FEniCS distribution and automatically run nightly (or more frequently) by the FEniCS buildbots. Their status is publicly available at http://fenicsproject.org/buildbot/.

5 Examples

In this section we provide some examples that cover the main aspects of solving PDEs on manifolds, and test results that demonstrate that our approach works. We have concentrated on the spherical case since that is the main surface of interest in geoscientific models. The example code is provided in the Supplement.

5.1 Two mixed formulations of Poisson's equation

In this section we discuss two different approaches to the discretisation of the gradient of the scalar solution u of a Poisson equation on the sphere. In the first approach, we use div-conforming finite element spaces for the vector field $\boldsymbol{\sigma} = \nabla u$ and rely upon the contravariant Piola transform to enforce tangency to the mesh used to approximate the sphere. In the second approach, we use Cartesian products of scalar finite element spaces (in this case discontinuous, piecewise polynomial spaces), and enforce approximate tangency through the introduction of Lagrange multipliers. This approach was advocated for fluid models on the sphere in Côté (1988) and used in conjunction with discontinuous Galerkin methods in Giraldo (2006).

In both of the following examples, we take Ω to be the surface of a unit sphere centred at the origin, and let $\mathcal{T}_h$ be an affine tessellation of this surface. For a given scalar function g, we seek the solution u of the Poisson equation written in *dual form*,

$$\boldsymbol{\sigma} - \nabla u = 0, \tag{38}$$

$$\mathrm{div}\boldsymbol{\sigma} + r = g, \tag{39}$$

where r is the domain average of g. Given a solution $(u, \boldsymbol{\sigma}, r)$, another solution can be obtained by adding an arbitrary constant to u; hence, we impose the condition

$$\int_\Omega u \mathrm{d}x = 0, \tag{40}$$

which fixes the value of this constant, leading to a unique solution.

5.1.1 Div-conforming spaces

To obtain a weak form of Eqs. (38) and (39), we consider the dot product of Eq. (38) with a vector-valued test function $\boldsymbol{\tau}$, integrated over the domain. Similarly, we multiply Eq. (39) by a scalar test function v and integrate over the domain, and multiply Eq. (40) by an arbitrary constant t. We apply integration by parts to transfer the derivative from u to $\boldsymbol{\tau}$ in the first equation. We obtain

$$\int_\Omega \boldsymbol{\sigma} \cdot \boldsymbol{\tau} \mathrm{d}x + \int_\Omega \mathrm{div}\boldsymbol{\tau} u \mathrm{d}x = 0, \tag{41}$$

$$\int_\Omega \mathrm{div}\boldsymbol{\sigma} v \mathrm{d}x + \int_\Omega r v \mathrm{d}x = \int_\Omega g v \mathrm{d}x, \tag{42}$$

$$\int_\Omega t u \mathrm{d}x = 0, \tag{43}$$

for all t and suitable[3] $(\boldsymbol{\tau}, v)$. Since $\boldsymbol{\tau}$, v and t are independent, we may combine these into a single equation:

$$\langle \boldsymbol{\sigma}, \boldsymbol{\tau} \rangle + \langle \mathrm{div}\boldsymbol{\sigma}, v \rangle + \langle \mathrm{div}\boldsymbol{\tau}, u \rangle + \langle r, v \rangle + \langle t, u \rangle = \langle g, v \rangle, \tag{44}$$

where we have adopted the angle bracket notation

$$\langle g v \rangle = \int_\Omega g v \mathrm{d}x, \tag{45}$$

for scalar variables (g, v) and

$$\langle \boldsymbol{\sigma}, \boldsymbol{\tau} \rangle = \int_\Omega \boldsymbol{\sigma} \cdot \boldsymbol{\tau} \mathrm{d}x, \tag{46}$$

for vector variables $(\boldsymbol{\sigma}, \boldsymbol{\tau})$.

To obtain a finite element discretisation of this discrete form, we simply restrict $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$ to a (vector-valued) finite element space V, and u and v to a different finite element space Q. It follows from inspection of Eq. (44) that V must be div-conforming, but that Q has no continuity constraints. It is well known in the literature, for example in Auricchio et al. (2004), that stable discretisations can be obtained when[4] the div operator maps from V onto Q; the loss of continuity means that Q may be a discontinuous finite element space. In this example we consider a number of such pairs of spaces that are available in FEniCS, presented in Table 1.

The problem was solved on a sequence of icosahedral meshes of the sphere, taking $g = x_1 x_2 x_3$. Example solutions and convergence plots are shown in Fig. 7, with the error measured using the L_2 norm $\|\cdot\|_0$ defined by

$$\|u\|_0 = \int_\Omega u^2 \mathrm{d}x. \tag{47}$$

As expected, we obtain first order convergence for $\mathrm{RT}_1 - \mathrm{DG}_0$ and $\mathrm{BDM}_1 - \mathrm{DG}_0$, and second order convergence for $\mathrm{BDFM}_2 - \mathrm{DG}_1$ and $\mathrm{BDM}_2 - \mathrm{DG}_1$. This example corroborates the veracity of the implementation of the contravariant Piola transformation on manifold meshes. The example code is provided in the Supplement in examples/mixed-poisson/hdiv-l2/mixed-poisson-sphere.py.

[3] "Suitable" meaning that the integrals are finite.

[4] Together with some easily satisfied technical conditions.

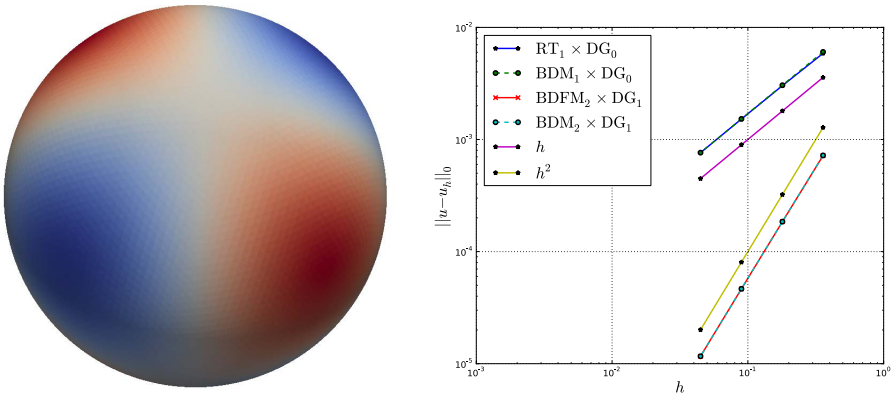

Fig. 7. Left: plot of the solution u to the div-conforming discretisation of the Poisson Eq. (44) with $V = \mathrm{RT}_1$ and $Q = \mathrm{DG}_0$. Right: $\|u - u_h\|_0$ versus mesh size h (where u is the exact solution and u_h is the numerical solution) for series of discretisations of Eq. (44).

Table 1. Triples (E, V, Q) of finite element spaces. Only the pairs (V, Q) are used in Sects. 5.1.1 and 5.2, while Sect. 5.5 also requires E. RT refers to the Raviart–Thomas space (Raviart and Thomas, 1977), BDM to the Brezzi–Douglas–Marini space (Brezzi et al., 1985), and BDFM to the Brezzi–Douglas–Fortin–Marini space (Brezzi and Fortin, 1991). Note that we have used the FEniCS numbering convention, in which the number refers to the highest order of polynomials appearing in the space, rather than the normal convention, in which the number reflects the order of numerical approximation; see Logg et al. (2012a) for further details. For example, in this numbering, the lowest order Raviart–Thomas space is denoted RT_1, rather than RT_0, whilst the BDFM space discussed in Cotter and Shipton (2012) is denoted BDFM_2, not BDFM_1. DG_0 and DG_1 denote discontinuous, piecewise constant and piecewise linear spaces, respectively. B_3 is the space spanned by cubic "bubble" functions that are nonzero only on a single element, and vanish on element boundaries.

E	V	Q
CG_1	RT_1	DG_0
CG_2	BDM_1	DG_0
$\mathrm{CG}_2 \oplus \mathrm{B}_3$	BDFM_2	DG_1
CG_3	BDM_2	DG_1

5.1.2 Cartesian product space with Lagrange multipliers

An alternative approach is to work with a Cartesian product finite element space, where each Cartesian component of the three-dimensional vector field $\boldsymbol{\sigma}$ is expanded in the same finite element space, and to enforce tangency of $\boldsymbol{\sigma}$ through a Lagrange multiplier, also expanded in the same space. With this approach, Eqs. (38) and (39) become

$$\boldsymbol{\sigma} - \nabla u - l\boldsymbol{k} = 0, \tag{48}$$

$$\mathrm{div}\boldsymbol{\sigma} + r = g, \tag{49}$$

$$\boldsymbol{\sigma} \cdot \boldsymbol{k} = 0, \tag{50}$$

where l is the Lagrange multiplier, and $\boldsymbol{k}$ is the unit outward normal to the manifold Ω. On a two-dimensional manifold, we introduce a Lagrange multiplier field $l \in \mathrm{DG}_k$; the finite element problem is to find $(\boldsymbol{\sigma}, u, l, r) \in W = \mathrm{DG}_k^3 \times \mathrm{CG}_{k+1} \times \mathrm{DG}_k \times \mathbb{R}$ such that

$$\langle \boldsymbol{\sigma}, \boldsymbol{\tau} \rangle - \langle \boldsymbol{\tau}, \nabla u \rangle + \langle \boldsymbol{\sigma}, \nabla v \rangle - \langle l, \boldsymbol{\tau} \cdot \boldsymbol{k} \rangle + \langle \gamma, \boldsymbol{\sigma} \cdot \boldsymbol{k} \rangle - \langle r, v \rangle + \langle t, u \rangle = -\langle g, v \rangle \tag{51}$$

for all $(\boldsymbol{\tau}, v, \gamma, t) \in W$.

Code for this example is provided in the Supplement in examples/mixed-poisson/l2-h1/mixed_poisson_l2_h1.py. Convergence plots are provided in Fig. 8. For the DG_0^3-CG_1 case, we observe second order convergence in the L_2-norm and first order convergence in the H_1-norm $\|\cdot\|_1$ defined by

$$\|u\|_1 = \int_\Omega u^2 + |\nabla u|^2 \mathrm{d}x, \tag{52}$$

in accordance with theory (Cotter et al., 2009).

We would usually expect these convergence rates to increase by one order when we change the spaces to $\mathrm{DG}_1^3 - \mathrm{CG}_2$. However, as discussed in (Bernard et al., 2008), higher-order convergence can only be achieved if higher-order approximations to the manifold itself are used, and in our implementation we use affine triangles. Hence, for $\mathrm{DG}_1^3 - \mathrm{CG}_2$, we also observe second and first order convergence for L_2 and H_1 norms, respectively. This example tests the use of three-dimensional vector fields on a two-dimensional manifold mesh.

5.2 Linear shallow water equations on the sphere

In this section, we use the framework to solve the linear shallow water equations in a rotating frame on the sphere. The unknowns are the depth of the shallow layer D and the velocity $\boldsymbol{u}$, assumed tangential to the sphere. They are related

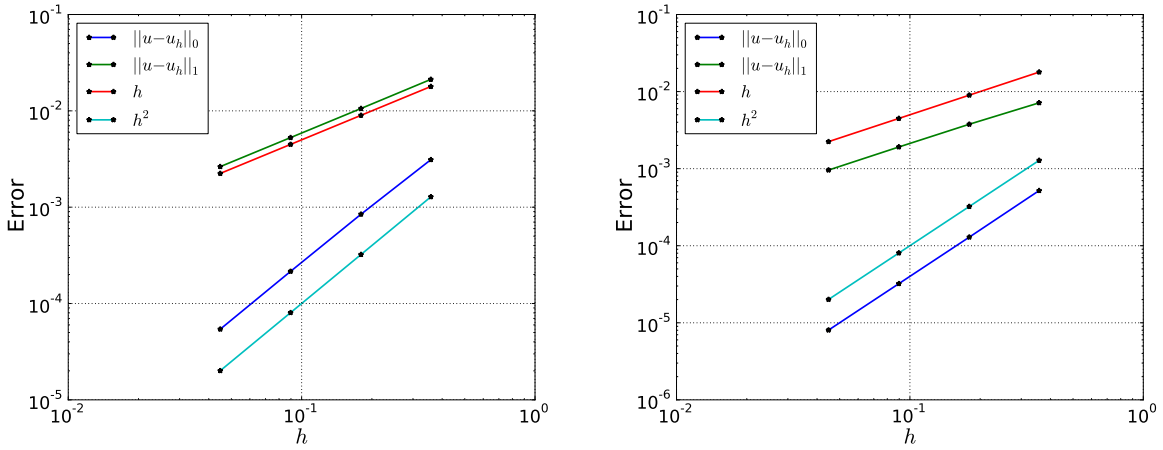

Fig. 8. $\|u-u_h\|_0$ and $\|u-u_h\|_1$ versus mesh size h (where u is the exact solution and u_h is the numerical solution of Eq. 51), using Lagrange multipliers to enforce approximate tangency. Left: DG_0^3-CG_1. Right: DG_1^3-CG_2.

by the equations:

$$\begin{aligned} \boldsymbol{u}_t + f\boldsymbol{u}^\perp + g\nabla D &= 0, \\ D_t + H\mathrm{div}\boldsymbol{u} &= 0, \end{aligned} \tag{53}$$

where g is the acceleration due to gravity; H is the (constant) reference layer depth; $f = \Omega_0 x_3/R$ is the Coriolis parameter, where Ω_0 is the rotation frequency of the sphere and R is the sphere radius; and $\boldsymbol{u}^\perp = \boldsymbol{k}\times\boldsymbol{u}$, where $\boldsymbol{k}$ is the unit normal vector to the sphere. The subscript t denotes the partial time derivative.

A weak form of these equations is obtained by taking the product with test functions $\boldsymbol{w}$ and ϕ, integrating over the domain Ω and integrating the gradient by parts. Restricting to finite element spaces V and Q with $\boldsymbol{w}, \boldsymbol{u} \in V$, and $D, \phi \in Q$, leads to the finite element discretisation

$$\langle \boldsymbol{w}, \boldsymbol{u}_t\rangle + \langle \boldsymbol{w}, f\boldsymbol{u}^\perp\rangle - \langle \mathrm{div}\boldsymbol{w}, gD\rangle = 0, \tag{54}$$

$$\langle \phi, D_t\rangle + \langle \phi, H\mathrm{div}\boldsymbol{u}\rangle = 0, \tag{55}$$

for all $\boldsymbol{w}$ in V and ϕ in Q. As discussed in Le Roux et al. (2005), it is important to choose a pair of finite element spaces for $\boldsymbol{u}$ and D that would be stable for the mixed Poisson problem (as described in Sect. 5.1), in order to avoid having spurious solutions where D is highly oscillatory in space but that have very slow frequencies in time. For large-scale atmosphere and ocean modelling, it is also important for the system to have exact steady state solutions in the f-plane (constant f) case corresponding to each divergence-free vector field in the finite element space for velocity. These solutions represent the large-scale balanced flow that slowly evolves in the nonlinear solutions, giving rise to "weather". It was shown in Cotter and Shipton (2012) that the stable element pairs using div-conforming elements for $\boldsymbol{u}$ such as those listed in Table 1 also satisfy this property, and hence we will use such spaces as examples here.

A direct computation shows that the total energy E_T given by

$$\begin{aligned} &E_T(t) = E_k(t) + E_p(t), \quad E_k(t) = 0.5H\|\boldsymbol{u}(t)\|_0^2, \\ &E_p(t) = 0.5g\|D(t)\|_0^2, \end{aligned} \tag{56}$$

is conserved for these spatial discretisations. It will also be exactly conserved by the implicit midpoint rule time discretisation method, which conserves all quadratic invariants (Leimkuhler and Reich, 2005, for example), and so we use this conservation as a diagnostic to verify our discretisation.

Snapshots of the solutions using the element combination $V = \mathrm{RT}_1$ and $Q = \mathrm{DG}_0$ are presented in Fig. 9. The computed energies for the same element combination are plotted in Fig. 10. We observe that the total energy is conserved (to within machine precision) as anticipated. Code for this example is provided in the Supplement in examples/linear-shallow-water/linear_shallow_water.py.

5.3 Linear wave equations on the torus

The Supplement also includes an example of the solution of the linear wave equation over a torus. The equations solved are Eqs. (54) and (55) with $f = 0$. The initial conditions provided are of a radially propagating wave. The results are qualitatively reasonable and energy conservation is observed to machine precision in a similar manner to that shown in Fig. 10. Since the problem solved does not have a straightforward analytic solution and is similar in character to the preceding sphere case, results are not reproduced in the paper. However this section serves as a pointer to the implementation in the Supplement for readers interested in simulating on manifolds other than the sphere. The code for this example is included in examples/torus. Of particular interest may

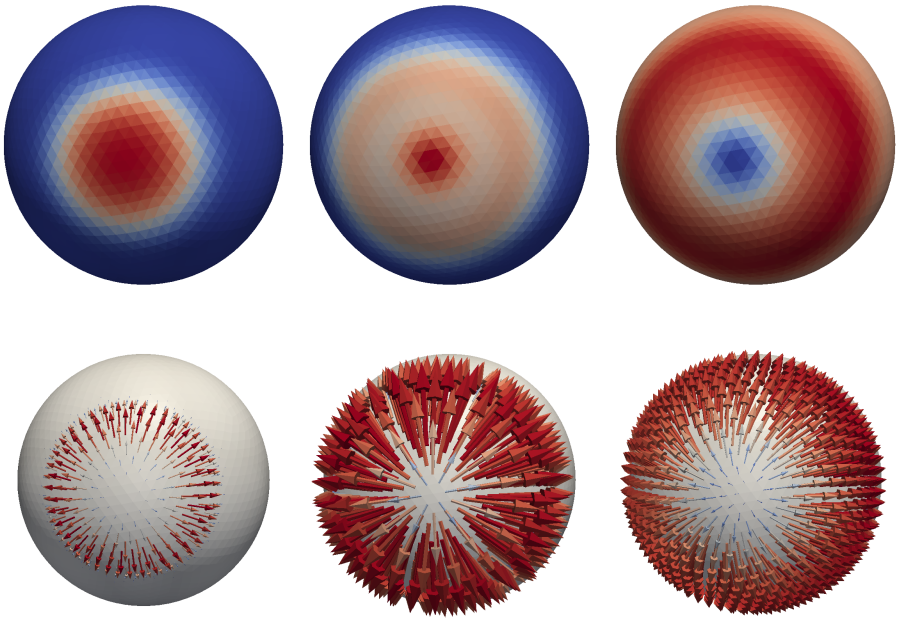

Fig. 9. Snapshots of the solution to the linear shallow water equations obtained using the method of Sect. 5.2, with $V = \mathrm{RT}_1$ and $Q = \mathrm{DG}_0$. Top: D. Bottom: $\boldsymbol{u}$.

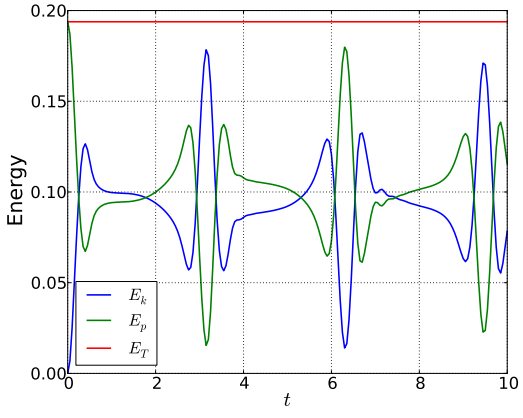

Fig. 10. Linear shallow water equations: kinetic E_k, potential E_p, and total energy E_T versus time t for an implicit midpoint in time and $\mathrm{RT}_1 \times \mathrm{DG}_0$ in space discretisation of Eqs. (54) and (55). As expected the total energy is conserved to machine precision (the maximum absolute conservation error is 1.4×10^{-15}).

be torus_mesh.py which contains a torus mesh object with the required specification of the global normal direction, and a facility for providing expressions such as initial conditions in manifold (polar) coordinates.

5.4 Upwind discontinuous Galerkin transport on the sphere

In this section we discuss the transport equation

$$D_t + \mathrm{div}(\boldsymbol{u} D) = 0, \tag{57}$$

which is a commonly encountered equation in the geosciences, describing the transport of a mass density D by a velocity $\boldsymbol{u}$; it appears in the shallow-water equations as the continuity equation describing the evolution of the layer depth D. This equation takes the form of a conservation law and therefore is ideally suited for discretisation using the discontinuous Galerkin (DG) approach which uses finite element spaces with no continuity constraints across element boundaries, and which extends the first order upwind finite volume method to higher order locally conservative schemes by increasing the order of the polynomials in each element.

To obtain the spatial discretisation, we multiply Eq. (57) by a discontinuous test function ϕ, integrate over a single cell T, and integrate by parts to obtain

$$\int_T \phi D_t \mathrm{d}x - \int_T \nabla\phi \cdot \boldsymbol{u} \mathrm{d}x + \int_{\partial T} \phi \tilde{D} \boldsymbol{u} \cdot \boldsymbol{n}_T \mathrm{d}s = 0, \tag{58}$$

where ∂T is the boundary of T; and $\boldsymbol{n}_T$ is the outward pointing normal vector to ∂T; and where $\tilde{D}$ is taken to be the value of D on the upwind side; that is, the side away from which the velocity $\boldsymbol{u}$, assumed continuous, is pointing. Local conservation follows from choosing $\phi = 1$ inside element T and $\phi = 0$ outside.

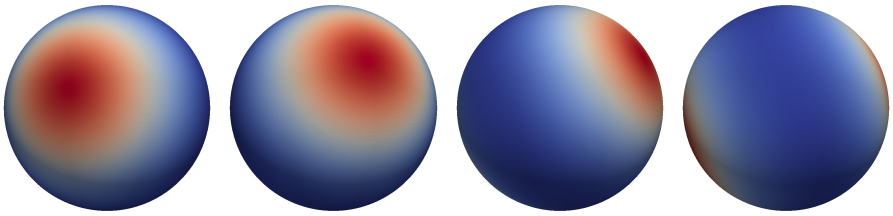

Fig. 11. Snapshots of the solution to the transport equation obtained using the method of Sect. 5.4.

To write this as a global system, we adopt an (arbitrary) global convention for labelling the two elements on each side of an interior facet: each interior facet $e = T^+ \cup T^-$, and we write ϕ^+ for $\phi|_{T^+}$ and ϕ^- for $\phi|_{T^-}$. We then sum Eq. (58) over all mesh elements and the problem becomes seek $D \in \mathrm{DG}_k$ such that

$$\int_\Omega \phi D_t \mathrm{d}x - \int_\Omega \nabla\phi \cdot \boldsymbol{u}\, D \mathrm{d}x + \int_\Gamma (\phi^+ - \phi^-) F \mathrm{d}s = 0, \qquad (59)$$

for all $\phi \in \mathrm{DG}_k$, where Γ is the union of all interior facets (here we have assumed for simplicity that Ω is closed so there is no $\partial\Omega$ contribution), and where we have introduced the flux $F = \tilde{D}\boldsymbol{u} \cdot \boldsymbol{n}^+$ (recall that $\boldsymbol{u} = \boldsymbol{u}^+$ by the assumption of continuous $\boldsymbol{u}$). Note that now an integral is performed over each facet only once, and so the Γ integrand contains contributions from both sides of each facet.

Finally, to express the equations in a form that can be easily written in UFL, we define the following function v over Γ:

$$v = \begin{cases} \boldsymbol{u} \cdot \boldsymbol{n} & \text{if } \boldsymbol{u} \cdot \boldsymbol{n} > 0 \\ 0 & \text{otherwise} \end{cases} = \frac{1}{2}(\boldsymbol{u} \cdot \boldsymbol{n} + |\boldsymbol{u} \cdot \boldsymbol{n}|). \qquad (60)$$

Then

$$F = v^+ D^+ - v^- D^-, \qquad (61)$$

and we write

$$\int_\Omega \phi D_t \mathrm{d}x - \int_\Omega \nabla\phi \cdot \boldsymbol{u}\, D \mathrm{d}x + \int_\Gamma (\phi^+ - \phi^-)(v^+ D^+ - v^- D^-) \mathrm{d}s = 0. \qquad (62)$$

The block diagonal structure of the mass matrix makes explicit methods attractive for DG, and strong stability preserving Runge–Kutta (SSPRK) methods are typically used, since they have usable Courant number restrictions for stability, and are shape-preserving when combined with suitable slope limiters (Cockburn and Shu, 2001). In this example, we use the third-order SSPRK method, without limiting.

We take as initial condition for D:

$$D_0 = e^{-(x_2^2 + x_3^2)}, \qquad (63)$$

where (x_1, x_2, x_3) are the global Cartesian coordinates as before, and we use the time-independent rigid rotation velocity field

$$\boldsymbol{u} = (-x_2, x_1, 0). \qquad (64)$$

This means that after integrating the equations from $t = 0$ until $t = 2\pi$, we recover the initial condition. For sufficiently small Δt, chosen so that spatial discretisation error is the dominant error term, the difference between D at $t = 2\pi$ and the initial conditions provides a metric for the spatial discretisation error.

Snapshots of the solutions using DG_1 are presented in Fig. 11. Moreover, plots of the L_2 error versus mesh size h are provided in Fig. 12, showing the expected second-order convergence. Note that higher order DG spaces would not yield a higher order convergence rate since we are using a second-order approximation to the sphere. This example tests the construction of facet normals and facet integrals on manifold meshes. Code for this example is provided in the Supplement in examples/dg-advection/dg-advection.py.

5.5 Nonlinear shallow water equations

The nonlinear shallow water equations are used to model a single incompressible thin layer of fluid with a free surface. They are often used as a test bed for horizontal discretisations for use in numerical weather prediction and ocean modelling. A linearised version was used previously in Sect. 5.2; the Coriolis parameter f and gravitational potential g are unchanged from before, but we allow spatial variations in topography, denoted by b. The velocity $\boldsymbol{u}$ and fluid depth D evolve according to

$$\boldsymbol{u}_t + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + f\boldsymbol{u}^\perp + g\nabla(D + b) = 0, \qquad (65)$$

$$D_t + \mathrm{div}(D\boldsymbol{u}) = 0. \qquad (66)$$

The momentum Eq. (65) can be rewritten in terms of the relative vorticity $\zeta = \nabla^\perp \cdot \boldsymbol{u} = \mathrm{div}(\boldsymbol{u} \times \boldsymbol{k})$, where $\boldsymbol{k}$ is again the local unit normal to the sphere:

$$\boldsymbol{u}_t + (\zeta + f)\boldsymbol{u}^\perp + \nabla\left(g(D + b) + \frac{1}{2}|\boldsymbol{u}|^2\right) = 0. \qquad (67)$$

Finally, defining a potential vorticity $q = \frac{\zeta+f}{D}$, we obtain the coupled equations

$$\boldsymbol{u}_t + q\,D\boldsymbol{u}^\perp + \nabla\left(g(D+b) + \frac{1}{2}|\boldsymbol{u}|^2\right) = 0, \tag{68}$$

$$D_t + \mathrm{div}(D\boldsymbol{u}) = 0. \tag{69}$$

We use a mixed finite element discretisation of this, stabilised with the Anticipated Potential Vorticity Method (APVM), which serves as a direct extension of the energy-conserving, enstrophy-dissipating C-grid finite difference scheme of Arakawa and Hsu (1990). We take $\boldsymbol{u} \in V$, $D \in Q$, where (V, Q) are chosen from the stable pairs of finite element spaces listed in Table 1. The spatially discretised equations are then

$$\langle \boldsymbol{w}, \boldsymbol{u}_t\rangle + \left\langle \boldsymbol{w}, (q - \underbrace{\tau(\boldsymbol{u}\cdot\nabla)q}_{\text{APVM term}})\boldsymbol{F}^\perp \right\rangle - \left\langle \mathrm{div}\boldsymbol{w}, g(D+b) + \frac{1}{2}|\boldsymbol{u}|^2\right\rangle = 0, \tag{70}$$

$$\langle \phi, D_t\rangle + \langle \phi, \mathrm{div}\boldsymbol{F}\rangle = 0, \tag{71}$$

for all $\boldsymbol{w}$ in V, ϕ in Q, where we have introduced a stabilisation parameter τ and the volume flux $\boldsymbol{F}$. Note that the potential vorticity $q \in E$ and the volume flux $\boldsymbol{F} \in V$ satisfy

$$\langle \gamma, q\,D\rangle = \left\langle -\nabla^\perp\gamma, \boldsymbol{u}\right\rangle + \langle \gamma, f\rangle, \tag{72}$$

$$\langle \boldsymbol{w}, \boldsymbol{F}\rangle = \langle \boldsymbol{w}, D\boldsymbol{u}\rangle, \tag{73}$$

for all γ in E and $\boldsymbol{w}$ in V. The finite element space E is chosen so that the $\nabla^\perp$ operator maps from E to V. Suitable choices of E, given V and Q, are also listed in Table 1.

To discretise in time, we will use the θ-method. Define

$$\boldsymbol{u}^\star = \boldsymbol{u}^n + (1-\theta)\Delta\boldsymbol{u}^n, \qquad D^\star = D^n + (1-\theta)\Delta D^n, \tag{74}$$

where $\Delta\boldsymbol{u}^n = \boldsymbol{u}^{n+1} - \boldsymbol{u}^n$, $\Delta D = D^{n+1} - D^n$. The fully discrete equations then read:

$$0 = \left\langle w, \Delta\boldsymbol{u}^n\right\rangle + \Delta t\left[\left\langle w, (q - \tau(\boldsymbol{u}^\star\cdot\nabla)q)\boldsymbol{F}^\perp\right\rangle - \left\langle \mathrm{div}\boldsymbol{w}, g(D^\star+b) + \frac{1}{2}|\boldsymbol{u}^\star|^2\right\rangle\right], \tag{75}$$

$$0 = \langle \phi, \Delta D^n\rangle + \Delta t\langle \phi, \mathrm{div}\boldsymbol{F}\rangle, \tag{76}$$

$$0 = \langle \gamma, q\,D^\star\rangle + \langle \nabla^\perp\gamma, \boldsymbol{u}^\star\rangle - \langle \gamma, f\rangle, \tag{77}$$

$$0 = \langle \boldsymbol{w}', \boldsymbol{F}\rangle - \langle \boldsymbol{w}', \boldsymbol{u}^\star D^\star\rangle, \tag{78}$$

for all $\boldsymbol{w}, \boldsymbol{w}' \in V$, $\phi \in Q$, $\gamma \in E$. We will take $\theta = \frac{1}{2}$ (implicit midpoint), and choose $\tau = \frac{1}{2}\Delta t$. A direct, "monolithic" approach is to solve Eq. (75) through Eq. (78) with DOLFIN's built-in nonlinear Newton-based solver. Example code implementing this approach is presented in examples/williamson2/auto.py.

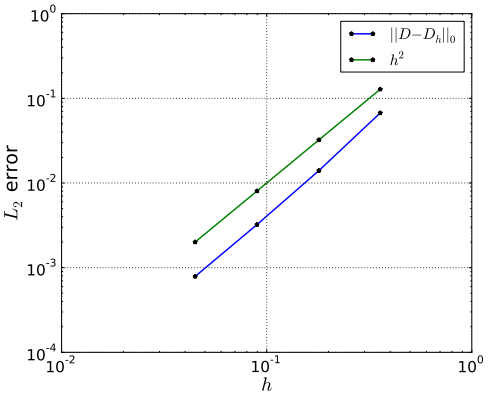

Fig. 12. $\|D - D_h\|_0$ versus mesh size h (where D is the exact solution and D_h is the numerical solution to the transport equation obtained using the method of Sect. 5.4). Second order convergence is observed as expected. Mass is conserved to machine precision (the maximum absolute conservation error is 1.7×10^{-14}).

The monolithic solver approach is somewhat inefficient, because the Newton iteration requires the solution of a large and difficult-to-precondition linear system. We shall now describe a more practical approach. Instead of treating q and $\boldsymbol{F}$ as prognostic variables to be solved for, we instead treat them as implicit functions of $\boldsymbol{u}$ and D, defined through Eqs. (77) and (78). With this definition, we then try to solve Eqs. (75) and (76) for $\Delta\boldsymbol{u}^n$ and ΔD^n. The difficulty is that it is no longer possible to use DOLFIN's built-in automatic differentiation to generate the Jacobian for this system. However, from physical considerations, the motion in each time step is dominated by the propagation of fast gravity waves; the potential vorticity q evolves on a much slower timescale. This means that we can approximate the Jacobian by treating q as if it is independent of $\boldsymbol{u}$ and D in Eq. (77) and by approximating $\boldsymbol{F}$ by $H\boldsymbol{u}$ in the Jacobian calculation, where H is the average of D over the domain. This motivates the use of the Jacobian from the linear shallow water equations Eqs. (54) and (55); this is the standard semi-implicit method for solving the shallow-water equations.

Since we have multiple Newton-like iterations within each time step, we will drop the time-dependent superscript n for clarity. We therefore state our problem at each time step as follows: given $\boldsymbol{u}$ and D, find $\Delta\boldsymbol{u}$, ΔD. Let $\Delta\boldsymbol{u}^k$ and ΔD^k be the approximations to $\Delta\boldsymbol{u}$ and ΔD obtained at the kth iteration. We aim to find a $\delta\boldsymbol{u}^{k+1}$ and δD^{k+1} with which to update $\Delta\boldsymbol{u}^k$ and ΔD^k:

$$\Delta\boldsymbol{u}^{k+1} = \Delta\boldsymbol{u}^k + \delta\boldsymbol{u}^{k+1}, \qquad \Delta D^{k+1} = \Delta D^k + \delta D^{k+1}. \tag{79}$$

Let $\boldsymbol{u}^\star = \boldsymbol{u}^n + (1-\theta)\Delta\boldsymbol{u}^k$, and similarly $D^\star$. We can then introduce $q^\star$ and $\boldsymbol{F}^\star$, satisfying

$$\langle \gamma, q^\star D^\star\rangle = -\langle \nabla^\perp\gamma, \boldsymbol{u}^\star\rangle + \langle \gamma, f\rangle \tag{80}$$

$$\langle \boldsymbol{w}', \boldsymbol{F}^\star\rangle = \langle \boldsymbol{w}', \boldsymbol{u}^\star D^\star\rangle \tag{81}$$

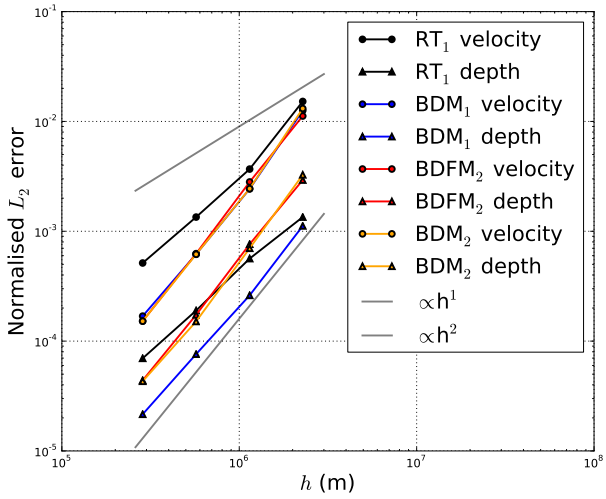

Fig. 13. Rigid rotation nonlinear shallow water test case on the sphere: L^2-error of the velocity $\boldsymbol{u}$ and depth D fields versus mesh size h.

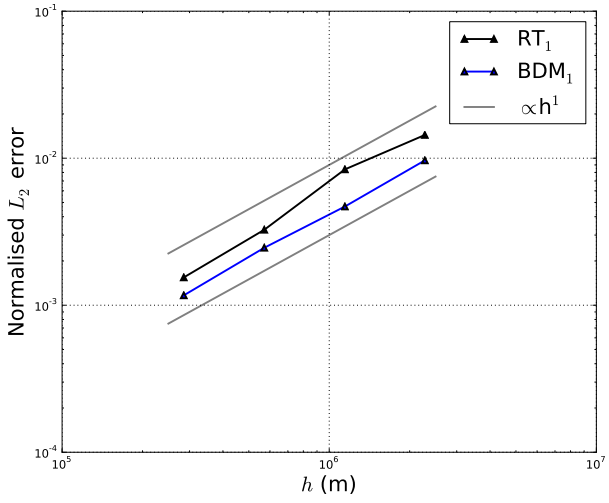

Fig. 14. Mountain nonlinear shallow water test case on the sphere: L^2-error of surface height field $D+b$ (compared to reference solution) versus mesh size h.

for all $\gamma \in E, \boldsymbol{w}' \in V$.

The equations for δu^{k+1} and δD^{k+1} are then

$$\langle \boldsymbol{w}, \delta \boldsymbol{u}^{k+1} \rangle + (1-\theta)\Delta t \left[\langle \boldsymbol{w}, f(\delta \boldsymbol{u}^{k+1})^{\perp} \rangle - \langle \mathrm{div}\boldsymbol{w}, g\delta D^{k+1} \rangle \right] = -\langle \boldsymbol{w}, \Delta \boldsymbol{u}^k \rangle - \Delta t \left[\langle \boldsymbol{w}, q^\star \boldsymbol{F}^{\star\perp} \rangle - \left\langle \mathrm{div}\boldsymbol{w}, gD^\star + \frac{1}{2}|\boldsymbol{u}^\star|^2 \right\rangle \right] \quad (82)$$

$$\langle \phi, \delta D^{k+1} \rangle + (1-\theta)\Delta t \langle \phi, H\mathrm{div}\delta \boldsymbol{u}^{k+1} \rangle = -\langle \phi, \Delta D^k \rangle - \Delta t \langle \phi, \mathrm{div}\boldsymbol{F}^\star \rangle \, . \quad (83)$$

It can be shown that, with the combinations of finite element spaces we employ, Eq. (83) implies that the equation

$$\delta D^{k+1} + (1-\theta)\Delta t H \mathrm{div}\delta \boldsymbol{u}^{k+1} = -\Delta D^k - \Delta t \mathrm{div}\boldsymbol{F}^\star \quad (84)$$

holds pointwise, and not just in an integral sense. We can therefore substitute for δD^{k+1} in the first equation and solve two separate equations, rather than a pair of coupled equations.

We illustrate this approach using two examples from the standard NCAR test set for shallow water equations on the sphere (Williamson et al., 1992), namely the solid rotation (test case 2) and mountain (test case 5) cases. The solid rotation case is an exact steady state solution of the nonlinear rotating shallow water equations, for which the velocity field is that of solid body rotation around the sphere. The metric for this test case is the L_2 norm of the difference from the initial conditions of the depth field D and the velocity $\boldsymbol{u}$ after five days. The fields were initialised by finite element projection into the relevant spaces having sampled the functions at quadrature points. Plots of the error for various finite element spaces are provided in Fig. 13, with the expected convergence rates. The mountain test case is a similar initial condition (with slightly different magnitude), but with a large conical mountain in the topography at mid-latitudes. This case does not have an analytical solution; the metric for this test is the L_2 norm of the difference between the surface height field $D+b$ and a high resolution reference solution obtained from the spectral model provided by NCAR, at 15 days. Plots of the error are provided in Fig. 14, and show the expected convergence rates. Figure 15 shows illustrative snapshots obtained from this simulation. Code illustrating the optimised approach applied to these two examples is presented in examples/williamson2/manual.py and examples/williamson5/w5.py. Note that energy conservation is not an expected property of the timestepping scheme employed for these tests, so no energy results are presented. A full analysis of the energy and enstrophy conservation properties of the spaces studied here is presented in McRae and Cotter (2013).

6 Limitations and extensions

The scope of the current implementation leaves room for a set of natural extensions.

First, the implementation only includes simplicial finite element cells and basis functions; that is, finite elements defined over intervals, triangles and tetrahedra. Moreover, only affine transformations from reference to physical cells are covered here. We remark that this is not due to a limitation in design: support for tensor product finite elements, including quadrilaterals and hexahedra, and curved cells is a natural extension and will be considered in future work. Note that for these cases, in contrast to the affine case, the Jacobian of the geometry transformation varies over each cell.

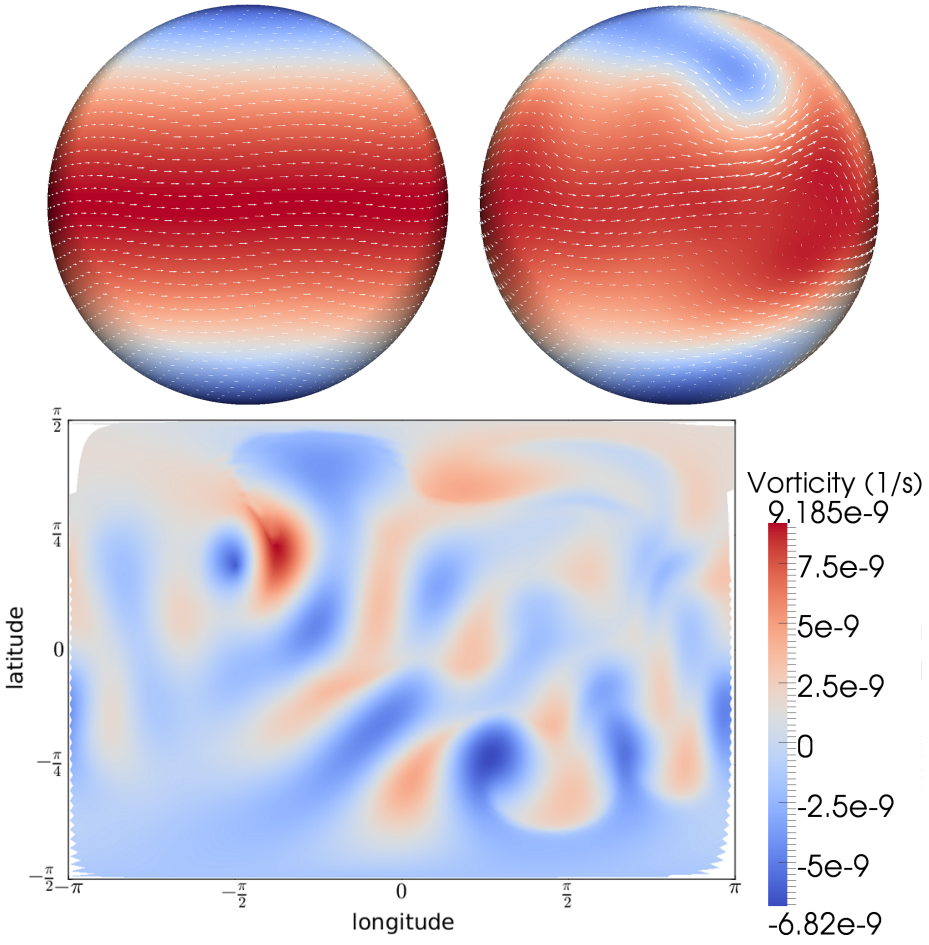

Fig. 15. Snapshots of the solution to test case 5 for the nonlinear shallow water equations at 15 days. These were obtained using the method of Sect. 5.5, with APVM stabilisation. We took $E = \mathrm{CG}_2 \oplus \mathrm{B}_3$, $V = \mathrm{BDFM}_2$ and $Q = \mathrm{DG}_1$. Top: initial surface height field, $D + b$, and velocity field, $\boldsymbol{u}$; final surface height and velocity fields. Bottom: final vorticity field, ζ, projected from a triangular mesh into latitude–longitude coordinates.

Second, we point out that the current UFL design assumes that mixed finite elements are defined in terms of a number of component elements sharing a common cell. A direct consequence of this is that mixed elements defined over different cells, for instance a mixed element with two components where one component is defined over cells of geometric dimension n and topological dimension m and another component is defined over cells of geometric dimension n and topological dimension $m - 1$, is not admitted. This restriction is however independent of the manifolds aspect: an extension of UFL for the case where $m = n$ would immediately carry over to the case $m < n$. Such an extension might be useful, for instance, in order to enable the imposition of a Lagrange multiplier over the surface of a mesh while solving an equation over the mesh as a whole.

7 Copyright and access to code

The FEniCS Project software, including the enhancements documented here, is available under version 3 of the GNU Lesser General Public License. The functionality described here is available in release version 1.2 and will be maintained in subsequent versions. FEniCS 1.2 consists of: DOLFIN 1.2.0, FFC 1.2.0, FIAT 1.1, Instant 1.2.0, UFC 2.2.0, UFL 1.2.0. Users are encouraged to employ the current release of FEniCS. This is available at http://fenicsproject.org/download. Archive packages for version 1.2 will remain available at http://fenicsproject.org/download/older_releases.html.

8 Supplementary material

The Supplement include the source code for the code examples given in Sect. 3 and for each of the numerical examples presented in Sect. 5. In particular, scripts are provided

to reproduce all of the graphs contained in this paper. Relevant references to the Supplement appear in the paper at the point at which the material is used, and further information is provided in the README file.

9 Conclusions

This paper details how the solution of finite element discretisations defined over simplicial meshes of immersed manifolds can be automated via code generation. The corresponding implementation is generally available as an integral part of FEniCS 1.2. The numerical examples presented cover a range of different partial differential equations and a wide range of different discretisations; we hope that these illustrate the flexibility and the strength of the approach and implementation.

Supplementary material related to this article is available online at http://www.geosci-model-dev.net/6/2099/2013/gmd-6-2099-2013-supplement.zip.

Acknowledgements. The development work in this paper was undertaken while the first three authors were visiting fellows at the Isaac Newton Institute programme on Multiscale Numerics for the Atmosphere and Ocean http://www.newton.ac.uk/programmes/AMM. The authors would like to thank the institute for its support and hospitality. The authors would also like to thank Hilary Weller for providing reference solutions for the mountain test case.

Marie E. Rognes is supported by a Center of Excellence grant from the Research Council of Norway to the Center for Biomedical Computing at Simula Research Laboratory.

Colin Cotter and David Ham acknowledge funding from the NERC Next Generation Weather and Climate Programme (NERC grants NE/I02013X/1 and NE/I021098/1). David Ham is also supported by EPSRC grant EP/I006079/1. Andrew McRae and David Ham acknowledge funding from the Grantham Institute for Climate Change.

Edited by: H. Weller

References

Alnæs, M. S.: UFL: a finite element form language, in: Automated Solution of Differential Equations by the Finite Element Method, vol. 84 of Lecture Notes in Computational Science and Engineering, edited by: Logg, A., Mardal, K.-A., and Wells, G. N., chap. 17, Springer, 2012.

Alnæs, M. S., Logg, A., Mardal, K.-A., Skavhaug, O., and Langtangen, H. P.: Unified Framework for Finite Element Assembly, Int. J. Computat. Sci. Eng., 4, 231–244, doi:10.1504/IJCSE.2009.029160, 2009.

Alnæs, M. S., Logg, A., and Mardal, K.-A.: UFC: a finite element code generation interface, in: Automated Solution of Differential Equations by the Finite Element Method, Volume 84 of Lecture Notes in Computational Science and Engineering, edited by: Logg, A., Mardal, K.-A., and Wells, G. N., chap. 16, Springer, 2012.

Alnæs, M. S., Logg, A., Ølgaard, K. B., Rognes, M. E., and Wells, G. N.: Unified Form Language: a domain-specific language for weak formulations of partial differential equations, ACM Trans. Mathe. Softw., available at: http://arxiv.org/abs/1211.4047 (last access: 13 December 2013), 2013.

Arakawa, A. and Hsu, Y.-J. G.: Energy conserving and potential-enstrophy dissipating schemes for the shallow water equations, Mon. Weather Rev., 118, 1960–1969, 1990.

Arakawa, A. and Lamb, V.: Computational design of the basic dynamical processes of the UCLA general circulation model, Meth. Computat. Phys., 17, 174–267, 1977.

Auricchio, F., Brezzi, F., and Lovadina, C.: Mixed Finite Element Methods, Encyclopedia of Computational Mechanics, 2004.

Barden, D. and Thomas, C.: An Introduction to Differential Manifolds, Imperial College Press, 2003.

Bernard, P.-E., Remacle, J.-F., and Legat, V.: High-order Discontinuous Galerkin Methods for Solving Conservation Laws on General 2-D Manifolds, 2008.

Brezzi, F. and Fortin, M.: Mixed and Hybrid Finite Element Methods, vol. 15 of Springer Series in Computational Mathematics, Springer-Verlag, New York, 1991.

Brezzi, F., Douglas, J., and Marini, L. D.: Two families of mixed finite elements for second order elliptic problems, Numer. Math., 47, 217–235, 1985.

Cockburn, B. and Shu, C.-W.: Runge–Kutta discontinuous Galerkin methods for convection-dominated problems, J. Sci. Comput., 16, 173–261, 2001.

Côté, J.: A Lagrange multiplier approach for the metric terms of semi-Lagrangian models on the sphere, Q. J. Roy. Meteorol. Soc., 114, 1347–1352, 1988.

Cotter, C. and Shipton, J.: Mixed finite elements for numerical weather prediction, J. Computat. Phys., 231, 7076–7091, doi:10.1016/j.jcp.2012.05.020, 2012.

Cotter, C. J., Ham, D. A., and Pain, C. C.: A mixed discontinuous/continuous finite element pair for shallow-water ocean modelling, Ocean Modell., 26, 86–90, doi:10.1016/j.ocemod.2008.09.002, 2009.

Dedner, A., Klöfkorn, R., Nolte, M., and Ohlberger, M.: A generic interface for parallel and adaptive discretization schemes: abstraction principles and the DUNE-FEM module, Computing, 90, 165–196, 2010.

DeSimone, A., Heltai, L., and Manigrasso, C.: Tools for the Solution of PDEs Defined on Curved Manifolds with deal.II, International School for Advanced Studies (SISSA), available at: http://hdl.handle.net/1963/3700 (last access: 13 December 2013), 2009.

Giraldo, F. X.: High-order triangle-based discontinuous Galerkin methods for hyperbolic equations on a rotating sphere, J. Computat. Phys., 214, 447–465, 2006.

Holm, D.: Geometric Mechanics – Part I: Dynamics and Symmetry, Imperial College Press, 2008.

Karniadakis, G. E. and Sherwin, S. J.: Spectral/hp Element Methods for CFD, Oxford University Press, 1999.

Kirby, R. C.: Algorithm 839: FIAT, a new paradigm for computing finite element basis functions, ACM T. Math. Software, 30, 502–516, doi:10.1145/1039813.1039820, 2004.

Kirby, R. C. and Logg, A.: A compiler for variational forms, ACM T. Math. Software, 32, 417–444, doi:10.1145/1163641.1163644, 2006.

Kuptsov, L.: Gram Determinant, in: Encyclopedia of Mathematics, Springer, available at: http://www.encyclopediaofmath.org/index.php/Gram_determinant?%oldid=18442 (last access: 2 July 2013), 2011.

Leimkuhler, B. and Reich, S.: Simulating Hamiltonian Dynamics, chap. 12, CUP, 2005.

Le Roux, D., Sène, A., Rostand, V., and Hanert, E.: On some spurious mode issues in shallow-water models using a linear algebra approach, Ocean Modell., 10, 83–94, 2005.

Logg, A. and Wells, G. N.: DOLFIN: automated finite element computing, ACM T. Math. Software, 37, 20:1–20:28, doi:10.1145/1731022.1731030, 2010.

Logg, A., Mardal, K.-A., and Wells, G.: Automated Solution of Differential Equations by the Finite Element Method: the Fenics Book, vol. 84, Springer, 2012a.

Logg, A., Mardal, K.-A., and Wells, G. N. (Eds.): Automated Solution of Differential Equations by the Finite Element Method, Springer, 2012b.

Logg, A., Mardal, K.-A., and Wells, G. N.: Finite element assembly, in: Automated Solution of Differential Equations by the Finite Element Method, Springer, 2012c.

Logg, A., Ølgaard, K. B., Rognes, M. E., and Wells, G. N.: FFC: the FEniCS Form Compiler, in: Automated Solution of Differential Equations by the Finite Element Method, vol. 84 of Lecture Notes in Computational Science and Engineering, edited by: Logg, A., Mardal, K.-A., and Wells, G. N., chap. 11, Springer, 2012d.

Logg, A., Wells, G. N., and Hake, J.: DOLFIN: a C++/Python Finite Element Library, in: Automated Solution of Differential Equations by the Finite Element Method, vol. 84 of Lecture Notes in Computational Science and Engineering, edited by: Logg, A., Mardal, K.-A., and Wells, G. N., chap. 10, Springer, 2012e.

McRae, A. T. T and Cotter, C. J.: Energy- and enstrophy-conserving schemes for the shallow-water equations, based on mimetic finite elements, Q. J. R. Meteorol. Soc., http://arxiv.org/abs/1305.4477 (last access: 13 December 2013), 2013.

Nédélec, J.-C.: Mixed finite elements in $\mathbb{R}^3$, Numer. Math., 35, 315–341, doi:10.1007/BF01396415, 1980.

Nédélec, J.-C.: A new family of mixed finite elements in $\mathbb{R}^3$, Numer. Math., 50, 57–81, 1986.

Penrose, R.: A generalized inverse for matrices, Proc. Cambridge Philos. Soc, 51, 406–413, 1955.

Raviart, P.-A. and Thomas, J. M.: A mixed finite element method for 2nd order elliptic problems, in: Mathematical aspects of finite element methods (Proc. Conf., Consiglio Naz. delle Ricerche (C.N.R.), Rome, 1975), 292–315, Lecture Notes in Math., vol. 606, Springer, Berlin, 1977.

Richardson, C. N., and Wells, G. N.: Expressive and scalable finite element simulation beyond 1000 cores, HECToR university distributed CSE project report, available at: http://www.hector.ac.uk/cse/distributedcse/reports/UniDOLFIN/, last access: 23 September 2013.

Rognes, M. E., Kirby, R. C., and Logg, A.: Efficient assembly of $H(\mathrm{div})$ and $H(\mathrm{curl})$ conforming finite elements, SIAM J. Sci. Comput., 31, 4130–4151, 2009.

Schmidt, A., Barth, T. J., and Siebert, K. G.: Design of adaptive finite element software: the finite element toolbox ALBERTA, vol. 42, Springer, 2005.

Sherwin, S., Kirby, R. M., and the Nektar++ team, available at: http://www.nektar.info, last access: 2 July 2013.

Williamson, D. L., Drake, J. B., Hack, J. J., Jakob, R., and Swarztrauber, P. N.: A standard test set for numerical approximations to the shallow water equations in spherical geometry, J. Computat. Phys., 102, 211–224, 1992.

Zienkiewicz, O. C., Taylor, R. L., and Zhu, J. Z.: The Finite Element Method: Its Basis and Fundamentals, Butterworth-Heinemann, 2005.