

Algorithms for pseudo-spectral reaction-diffusion solver on evolving surfaces

1 Introduction

This note gives a specification of the three algorithms used to evaluate the Laplace–Beltrami operator on a closed surface $\Gamma \subset \mathbb{R}^3$, assumed to be a smooth deformation of the unit sphere.

2 Notation and setup

Points of the unit sphere $\mathbb{S}^2$ are parametrized by colatitude $\theta \in [0, \pi]$ and longitude $\varphi \in [0, 2\pi)$. $\theta = 0$ denotes the North pole, and $\theta = \pi$ denotes the South pole. Assuming the surface $\Gamma \subset \mathbb{R}^3$ is a smooth deformation of the unit sphere, we can represent Γ as a smooth embedding $X : \mathbb{S}^2 \rightarrow \mathbb{R}^3$, $X(\theta, \varphi) = (x(\theta, \varphi), y(\theta, \varphi), z(\theta, \varphi))$. The individual coordinates of the embedding x, y, z , are each smooth functions on the unit sphere. We choose to represent the surface via its expansion in spherical harmonics:

$$X(\theta, \varphi) = \begin{pmatrix} x(\theta, \varphi) \\ y(\theta, \varphi) \\ z(\theta, \varphi) \end{pmatrix} = \sum_{\ell=0}^L \sum_{m=-\ell}^{\ell} \begin{pmatrix} \hat{x}_{\ell}^m \\ \hat{y}_{\ell}^m \\ \hat{z}_{\ell}^m \end{pmatrix} Y_{\ell}^m(\theta, \varphi) = \sum_{\ell=0}^L \sum_{m=-\ell}^{\ell} \hat{X}_{\ell}^m Y_{\ell}^m(\theta, \varphi).$$

$Y_{\ell}^m(\theta, \varphi)$ denote the $\mathbb{S}^2$ -orthonormal spherical harmonics (with Condon–Shortley phase.) In particular, we represent Γ through the coefficients $\hat{X}_{\ell}^m$. A scalar field $u : \Gamma \rightarrow \mathbb{R}$ can also be represented by spherical harmonic coefficients:

$$u(\theta, \varphi) = \sum_{\ell=0}^L \sum_{m=-\ell}^{\ell} u_{\ell}^m Y_{\ell}^m(\theta, \varphi). \quad (2.1)$$

Note that, because the functions u, x, y , and z are real-valued the coefficients obey a conjugate symmetry $u_{\ell}^{-m} = (-1)^m \overline{u_{\ell}^m}$, so only the coefficients with $m \geq 0$ need be stored. Throughout the computation, we transform between two representations of u :

- **Coefficient space:** the array $\{u_{\ell}^m\}$ above.
- **Grid space:** the values of u at a tensor-product set of nodes $\{(\theta_i, \varphi_j)\}$ – e.g. Gauss–Legendre nodes in $\cos \theta$ crossed with an equally-spaced grid in φ .

Passing between the two is done by

- *Synthesis* $\mathcal{S}$: coefficients $\rightarrow$ grid values, i.e. evaluating eq. (2.1) (and its real-field completion) at the grid nodes;
- *Analysis* $\mathcal{A}$: grid values $\rightarrow$ coefficients, i.e. numerically evaluating the projection integral $u_{\ell}^m = \int_{\mathbb{S}^2} u \overline{Y_{\ell}^m} d\Omega$ by quadrature on the grid.

2.1 Differentiating Y_{ℓ}^m

In section 4, we will need to differentiate the coordinates of our embedding X with respect to θ and φ . Differentiating the coordinate maps requires taking partial derivatives of spherical harmonics, which we describe here.

We use the following formulae:

$$\frac{\partial Y_\ell^m}{\partial \varphi} = im Y_\ell^m, \quad (2.2)$$

$$\frac{\partial^2 Y_\ell^m}{\partial \varphi^2} = -m^2 Y_\ell^m, \quad (2.3)$$

$$\sin(\theta) \frac{\partial Y_\ell^m}{\partial \theta} = \alpha^+(\ell, m) Y_{\ell+1}^m + \alpha^-(\ell, m) Y_{\ell-1}^m, \quad (2.4)$$

$$\sin^2(\theta) \frac{\partial^2 Y_\ell^m}{\partial \theta^2} = \beta^+(\ell, m) Y_{\ell+2}^m + \beta^0(\ell, m) Y_\ell^m + \beta^-(\ell, m) Y_{\ell-2}^m, \quad (2.5)$$

In eqs. (2.4) and (2.5), any term referring to a degree outside $0 \leq \ell \leq L$, or to $|m| > \ell$ is dropped. The coefficients $\alpha^+, \alpha^-, \beta^+, \beta^0, \beta^-$ come from recurrence relations of associated Legendre polynomials. Writing $\mathcal{L} = \ell(\ell + 1)$, we have

$$\alpha^+(\ell, m) = \ell \sqrt{\frac{(\ell - m + 1)(\ell + m + 1)}{(2\ell + 1)(2\ell + 3)}}, \quad (2.6)$$

$$\alpha^-(\ell, m) = -(\ell + 1) \sqrt{\frac{(\ell - m)(\ell + m)}{(2\ell - 1)(2\ell + 1)}}, \quad (2.7)$$

$$\beta^+(\ell, m) = \frac{\ell^2}{2\ell + 3} \sqrt{\frac{(\ell - m + 1)(\ell - m + 2)(\ell + m + 1)(\ell + m + 2)}{(2\ell + 1)(2\ell + 5)}}, \quad (2.8)$$

$$\beta^0(\ell, m) = \frac{-2\mathcal{L}^2 + (3 + 2m^2)\mathcal{L} - 6m^2}{(2\ell + 3)(2\ell - 1)}, \quad (2.9)$$

$$\beta^-(\ell, m) = \frac{(\ell + 1)^2}{2\ell - 1} \sqrt{\frac{(\ell + m)(\ell + m - 1)(\ell - m)(\ell - m - 1)}{(2\ell + 1)(2\ell - 3)}}. \quad (2.10)$$

These coefficients can be precomputed once for all (ℓ, m) with $0 \leq m \leq \ell \leq L$.

3 Computing partial derivatives of $u(\theta, \varphi)$

Given the coefficients $\{u_\ell^m\}$ of a scalar field $u(\theta, \varphi)$, computing $\partial_\varphi u$ and $\partial_\varphi^2 u$ is simple. Computing $\partial_\theta u$ and $\partial_\theta^2 u$ requires two steps. First, we use eqs. (2.4) and (2.5) termwise to compute v_ℓ^m , the coefficients of $\sin(\theta) \partial_\theta u$ and w_ℓ^m , the coefficients of $\sin^2(\theta) \partial_\theta^2 u$:

$$v_\ell^m = \alpha^+(\ell - 1, m) u_{\ell-1}^m + \alpha^-(\ell + 1, m) u_{\ell+1}^m, \quad (3.1)$$

$$w_\ell^m = \beta^+(\ell - 2, m) u_{\ell-2}^m + \beta^0(\ell, m) u_\ell^m + \beta^-(\ell + 2, m) u_{\ell+2}^m, \quad (3.2)$$

(again dropping any out-of-range term). We then transform to gridspace and divide by $\sin(\theta)$ or $\sin^2(\theta)$. The full procedure for computing partial derivatives of u is described in Algorithm 1.

3.1 Locations in the Python code

The routines for computing first partial derivatives and the α coefficients are in `src/surface_gradient/partial_derivatives`. The routines for computing second derivatives and the β coefficients are in `src/surface_gradient/second_partial_derivatives`.

4 Computing geometric quantities on Γ

4.1 Metric and inverse metric quantities

Recall from section 2 that Γ is given by the embedding $X(\theta, \varphi) = (x(\theta, \varphi), y(\theta, \varphi), z(\theta, \varphi))$, with each Cartesian component represented in coefficient space exactly as in section 2, up to the same

Algorithm 1: Computing partial derivatives of $u(\theta, \varphi)$

Input : Coefficients $\{u_\ell^m\}$, $0 \leq m \leq \ell \leq L$; grid nodes $\{(\theta_i, \varphi_j)\}$; precomputed recurrence coefficients $\alpha^\pm, \beta^{+,0,-}$.

Output: Grid space values of $\partial_\theta u$, $\partial_\varphi u$, $\partial_\theta^2 u$, $\partial_\varphi^2 u$, $\partial_\theta \partial_\varphi u$.

```

//  $\varphi$ -derivatives
1  $(\partial_\varphi u)_\ell^m \leftarrow im u_\ell^m$  for all  $(\ell, m)$ 
2  $(\partial_\varphi^2 u)_\ell^m \leftarrow -m^2 u_\ell^m$  for all  $(\ell, m)$ 
3  $\partial_\varphi u \leftarrow \mathcal{S}((\partial_\varphi u)_\ell^m)$ ,  $\partial_\varphi^2 u \leftarrow \mathcal{S}((\partial_\varphi^2 u)_\ell^m)$  // transform to grid space

//  $\theta$ -derivative, via eq. (3.1)
4  $v_\ell^m \leftarrow \alpha^+(\ell-1, m)u_{\ell-1}^m + \alpha^-(\ell+1, m)u_{\ell+1}^m$  for all  $(\ell, m)$ 
5  $\partial_\theta u \leftarrow \mathcal{S}(v_\ell^m) / \sin \theta$  // division is elementwise, on the grid

//  $\theta\theta$ -derivative, via eq. (3.2)
6  $w_\ell^m \leftarrow \beta^+(\ell-2, m)u_{\ell-2}^m + \beta^0(\ell, m)u_\ell^m + \beta^-(\ell+2, m)u_{\ell+2}^m$  for all  $(\ell, m)$ 
7  $\partial_\theta^2 u \leftarrow \mathcal{S}(w_\ell^m) / \sin^2 \theta$ 

// mixed derivative: apply  $im$  to the  $v_\ell^m$  already computed
8  $\partial_\theta \partial_\varphi u \leftarrow \mathcal{S}(im v_\ell^m) / \sin \theta$ 

```

truncation degree L . The tangent vectors to Γ at (θ, φ) are

$$X_\theta = \frac{\partial X}{\partial \theta}, \quad X_\varphi = \frac{\partial X}{\partial \varphi},$$

each computed componentwise using Algorithm 1. The first fundamental form (metric tensor) of Γ in the (θ, φ) coordinates has components

$$g = \begin{bmatrix} g_{\theta\theta} & g_{\theta\varphi} \\ g_{\varphi\theta} & g_{\varphi\varphi} \end{bmatrix} = \begin{bmatrix} X_\theta \cdot X_\theta & X_\theta \cdot X_\varphi \\ X_\varphi \cdot X_\theta & X_\varphi \cdot X_\varphi \end{bmatrix}, \quad (4.1)$$

evaluated pointwise on the grid. We can use a closed-form expression for the inverse of a 2×2 matrix to find g^{-1} :

$$g^{-1} = \begin{bmatrix} g^{\theta\theta} & g^{\theta\varphi} \\ g^{\varphi\theta} & g^{\varphi\varphi} \end{bmatrix} = \frac{1}{\det(g)} \begin{bmatrix} g_{\varphi\varphi} & -g_{\theta\varphi} \\ -g_{\varphi\theta} & g_{\theta\theta} \end{bmatrix}. \quad (4.2)$$

The surface gradient is defined as

$$\nabla_\Gamma u = \begin{pmatrix} \frac{\partial_x^\Gamma u}{\partial_z^\Gamma u} \end{pmatrix} = \underbrace{\begin{bmatrix} X_\theta & X_\varphi \end{bmatrix} g^{-1}}_{=[V_\theta \quad V_\varphi]} \begin{bmatrix} \partial_\theta u \\ \partial_\varphi u \end{bmatrix}.$$

The contraction between the tangent vectors and the inverse metric tensor are called the “inverse metric quantities.” They only depend on the surface Γ , and not the function u being differentiated, so we precompute and store them.

$$V_\theta := g^{\theta\theta} X_\theta + g^{\theta\varphi} X_\varphi, \quad V_\varphi := g^{\varphi\theta} X_\theta + g^{\varphi\varphi} X_\varphi. \quad (4.3)$$

Algorithm 2 describes how these quantities are computed.

4.2 Normal vectors and mean curvature

Measuring how Γ curves in $\mathbb{R}^3$ requires the second partial derivatives $X_{\theta\theta}, X_{\theta\varphi}, X_{\varphi\varphi}$ of the embedding, each computed componentwise via Algorithm 1, together with an outward unit normal.

Algorithm 2: Computing inverse metric quantities of Γ

Input : Coefficients of the embedding components x, y, z .

Output: Inverse metric quantities $V_\theta, V_\varphi \in \mathbb{R}^3$.

```
// Tangent vectors, via Algorithm 1 applied to each of  $x, y, z$ 
1  $X_\theta \leftarrow (\partial_\theta x, \partial_\theta y, \partial_\theta z)$ 
2  $X_\varphi \leftarrow (\partial_\varphi x, \partial_\varphi y, \partial_\varphi z)$ 

// First fundamental form, eq. (4.1), in grid space
3  $g_{\theta\theta} \leftarrow X_\theta \cdot X_\theta, \quad g_{\theta\varphi} \leftarrow X_\theta \cdot X_\varphi, \quad g_{\varphi\varphi} \leftarrow X_\varphi \cdot X_\varphi$ 
4  $\det(g) \leftarrow g_{\theta\theta}g_{\varphi\varphi} - g_{\theta\varphi}^2$ 

// Inverse metric, eq. (4.2), in grid space
5  $g^{\theta\theta} \leftarrow g_{\varphi\varphi}/\det(g), \quad g^{\varphi\varphi} \leftarrow g_{\theta\theta}/\det(g), \quad g^{\theta\varphi} \leftarrow -g_{\theta\varphi}/\det(g)$ 

// Inverse metric quantities, eq. (4.3), in grid space
6  $V_\theta \leftarrow g^{\theta\theta}X_\theta + g^{\theta\varphi}X_\varphi$ 
7  $V_\varphi \leftarrow g^{\varphi\theta}X_\theta + g^{\varphi\varphi}X_\varphi$ 
```

An unnormalized normal vector at each point is given by the cross product of the tangent vectors,

$$\tilde{N} := X_\varphi \times X_\theta.$$

Since $|X_\varphi \times X_\theta|^2 = (X_\varphi \cdot X_\varphi)(X_\theta \cdot X_\theta) - (X_\varphi \cdot X_\theta)^2 = \det(g)$, the unit normal is

$$n = \frac{\tilde{N}}{\sqrt{\det(g)}} = \frac{X_\varphi \times X_\theta}{\sqrt{\det(g)}},$$

evaluated pointwise in grid space.

The second fundamental form of Γ collects the normal components of the second derivatives of X :

$$b_{\theta\theta} = X_{\theta\theta} \cdot n, \quad b_{\theta\varphi} = b_{\varphi\theta} = X_{\theta\varphi} \cdot n, \quad b_{\varphi\varphi} = X_{\varphi\varphi} \cdot n. \quad (4.4)$$

The mean curvature H (the average of the two principal curvatures) is half the trace of the shape operator $g^{-1}b$,

$$H = \frac{1}{2} \operatorname{tr}(g^{-1}b) = \frac{1}{2} (g^{\theta\theta}b_{\theta\theta} + 2g^{\theta\varphi}b_{\theta\varphi} + g^{\varphi\varphi}b_{\varphi\varphi}), \quad (4.5)$$

with $g^{\theta\theta}, g^{\theta\varphi}, g^{\varphi\varphi}$ the inverse metric components of eq. (4.2). Algorithm 3 describes how n and H are computed.

4.3 Locations in the Python code

The code implements a `SurfaceDiffOperator` class in `src/surface_gradient/SurfaceDiffOperator.py` which is initialized with coefficients of the x, y, z embedding and computes the geometric quantities. Upon initialization, these classmethods are called:

- `SurfaceDiffOperator._precompute_metric_quantities()` which computes V_θ and V_φ .
- `SurfaceDiffOperator._precompute_curvature()` which (optionally) computes the quantities in section 4.2.

5 Evaluating the surface Laplace-Beltrami operator $\Delta_\Gamma u$

Let u be a scalar field on Γ , pulled back to $\mathbb{S}^2$ via X and represented in coefficient space as in Section 2. Using the inverse metric, the surface (tangential) gradient of u , expressed as an $\mathbb{R}^3$ -

Algorithm 3: Computing the unit normal and mean curvature of Γ

Input : Coefficients of the embedding components x, y, z ; inverse metric $\det(g), g^{\theta\theta}, g^{\theta\varphi}, g^{\varphi\varphi}$.

Output: Unit normal $n \in \mathbb{R}^3$ and mean curvature H , in grid space.

```
// First and second derivatives of the embedding, via Algorithm 1 applied to
// each of  $x, y, z$ 
1  $X_\theta \leftarrow (\partial_\theta x, \partial_\theta y, \partial_\theta z)$ 
2  $X_\varphi \leftarrow (\partial_\varphi x, \partial_\varphi y, \partial_\varphi z)$ 
3  $X_{\theta\theta} \leftarrow (\partial_\theta^2 x, \partial_\theta^2 y, \partial_\theta^2 z)$ 
4  $X_{\theta\varphi} \leftarrow (\partial_\theta \partial_\varphi x, \partial_\theta \partial_\varphi y, \partial_\theta \partial_\varphi z)$ 
5  $X_{\varphi\varphi} \leftarrow (\partial_\varphi^2 x, \partial_\varphi^2 y, \partial_\varphi^2 z)$ 

// Unit normal
6  $\tilde{N} \leftarrow X_\varphi \times X_\theta$ 
7  $n \leftarrow \tilde{N} / \sqrt{\det(g)}$  // elementwise, on the grid

// Second fundamental form, eq. (4.4)
8  $b_{\theta\theta} \leftarrow X_{\theta\theta} \cdot n, \quad b_{\theta\varphi} \leftarrow X_{\theta\varphi} \cdot n, \quad b_{\varphi\varphi} \leftarrow X_{\varphi\varphi} \cdot n$ 

// Mean curvature, eq. (4.5)
9  $H \leftarrow \frac{1}{2}(g^{\theta\theta}b_{\theta\theta} + 2g^{\theta\varphi}b_{\theta\varphi} + g^{\varphi\varphi}b_{\varphi\varphi})$ 
```

valued field along Γ , is

$$\nabla_\Gamma u = \underbrace{\begin{bmatrix} X_\theta & X_\varphi \end{bmatrix} g^{-1}}_{=[V_\theta \quad V_\varphi]} \begin{bmatrix} \partial_\theta u \\ \partial_\varphi u \end{bmatrix}, \quad (5.1)$$

with V_θ, V_φ as in eq. (4.3). This vector field is tangent to Γ everywhere (it lies in $\text{span}\{X_\theta, X_\varphi\}$ at each point). For any $\mathbb{R}^3$ -valued field $F(\theta, \varphi) = (F_x, F_y, F_z)$ along Γ that is tangent to Γ (such as $\nabla_\Gamma u$), the surface divergence is likewise expressed through the inverse metric quantities,

$$\text{div}_\Gamma F = V_\theta \cdot \frac{\partial F}{\partial \theta} + V_\varphi \cdot \frac{\partial F}{\partial \varphi} = \sum_{i \in \{x, y, z\}} \left(V_{\theta, i} \frac{\partial F_i}{\partial \theta} + V_{\varphi, i} \frac{\partial F_i}{\partial \varphi} \right). \quad (5.2)$$

The surface Laplace–Beltrami operator is then simply the composition

$$\Delta_\Gamma u = \text{div}_\Gamma (\nabla_\Gamma u). \quad (5.3)$$

Evaluating eqs. (5.1) and (5.2) numerically requires moving the three Cartesian components of $\nabla_\Gamma u$ back into coefficient space (via analysis $\mathcal{A}$) so that Algorithm 1 can differentiate them again. This procedure is described in Algorithm 4.

5.1 Locations in the Python code

Here are important locations in the code:

- `src/surface_screened_laplacian::surface_screened_laplacian()` computes $v \mapsto (I + c\Delta_\Gamma)v$. The identity term and coefficient c will be explained in the following section.
- There `SurfaceDiffOperator` class also has classmethods implementing Δ_Γ and $(I + c\Delta_\Gamma)$. These should be treated as deprecated.

Algorithm 4: Evaluating the surface Laplace-Beltrami operator $\Delta_\Gamma u$

Input : Coefficients $\{u_\ell^m\}$ of u ; inverse metric quantities V_θ, V_φ .

Output: Coefficients $\{(\Delta_\Gamma u)_\ell^m\}$ of $\Delta_\Gamma u$.

// Surface gradient, eq. (5.1)

1 Compute $\partial_\theta u, \partial_\varphi u$ on the grid (Algorithm 1)

2 **for** $i \in \{x, y, z\}$ **do**

3 $(\nabla_\Gamma u)_i \leftarrow \partial_\theta u \cdot V_{\theta,i} + \partial_\varphi u \cdot V_{\varphi,i}$ // grid space

// Move each component back to coefficient space to differentiate again

4 **for** $i \in \{x, y, z\}$ **do**

5 $(\nabla_\Gamma u)_i^{\ell m} \leftarrow \mathcal{A}((\nabla_\Gamma u)_i)$

6 Compute $\partial_\theta (\nabla_\Gamma u)_i, \partial_\varphi (\nabla_\Gamma u)_i$ on the grid (Algorithm 1)

// Surface divergence, eq. (5.2)

7 $\Delta_\Gamma u \leftarrow \sum_{i \in \{x, y, z\}} \left(V_{\theta,i} \partial_\theta (\nabla_\Gamma u)_i + V_{\varphi,i} \partial_\varphi (\nabla_\Gamma u)_i \right)$ // grid space

8 $\{(\Delta_\Gamma u)_\ell^m\} \leftarrow \mathcal{A}(\Delta_\Gamma u)$

6 Implicit timestepping and the linear solve

As a motivating example, consider the heat equation on Γ , that is, finding a solution $u(x, t) : \Gamma \times [0, T] \rightarrow \mathbb{R}$ such that

$$\frac{\partial u}{\partial t} = \Delta_\Gamma u, \quad (6.1)$$

with initial condition $u(x, 0) = u_0$ given. We describe how one step of an implicit time discretization of eq. (6.1) is solved, using Algorithm 4 as a subroutine together with a preconditioned GMRES solve.

6.1 Backward Euler time discretization

We discretize eq. (6.1) in time with the backward Euler method: given the current solution u^n (coefficients, as in section 2) and a timestep Δt , the new solution u^{n+1} solves

$$\underbrace{u^{n+1} - \Delta t \Delta_\Gamma u^{n+1}}_{=Au^{n+1}} = u^n. \quad (6.2)$$

This equation is linear in the unknown u^{n+1} . We will compute a solution by defining the action of $A = (I - \Delta t \Delta_\Gamma)$ via its matrix-vector product and using GMRES to solve the linear system.

6.2 Preconditioning via the round-sphere Laplacian

We choose to precondition the problem using the operator

$$M = I - \Delta t \Delta_{\mathbb{S}^2}$$

as our preconditioner. $\Delta_{\mathbb{S}^2}$ is the Laplace-Beltrami operator on the unit sphere; it is diagonal in coefficient space:

$$\Delta_{\mathbb{S}^2} Y_\ell^m = -\ell(\ell + 1) Y_\ell^m. \quad (6.3)$$

This means M^{-1} is diagonal in coefficient space. Consequently M^{-1} is applied by elementwise division,

$$(M^{-1}v)_\ell^m = \frac{v_\ell^m}{1 + \Delta t \ell(\ell + 1)}. \quad (6.4)$$

6.3 Real vector space embedding

Because we are working with real-valued functions u , only coefficients with $m \geq 0$ are stored, and the $m = 0$ coefficients must have 0 imaginary part (section 2). The coefficient array $u_\ell^m \in \mathbb{C}^d$ is not closed under arbitrary complex linear combinations, so it is not, strictly, a complex vector space. It is, however, a genuine real vector space of dimension $(L+1)^2$: one real degree of freedom per $m = 0$ mode ($L+1$ of these), plus one real and one imaginary degree of freedom per $m > 0$ mode. Before running GMRES, we map the half-spectrum complex array to this real vector space via the fixed linear bijection

$$\mathcal{E}(u) := \left((\Re(u_\ell^0))_{\ell=0}^L, (\Re(u_\ell^m))_{0 < m \leq \ell \leq L}, (\Im(u_\ell^m))_{0 < m \leq \ell \leq L} \right) \in \mathbb{R}^{(L+1)^2}, \quad (6.5)$$

i.e. concatenating the real parts of the $m = 0$ coefficients, the real parts of the $m > 0$ coefficients, and the imaginary parts of the $m > 0$ coefficients.

6.4 Miscellaneous implementation details

These are miscellaneous details about the implementation of the operator $A = (I - \Delta t \Delta_I)$.

- A sets $\ell \geq L - 2$ coefficients to 0; this is because the recurrence formulae in Section 2 can exactly represent partial derivatives when $\ell < L - 2$.

6.5 Solving the implicit step with GMRES

We solve for u^{n+1} with a standard restarted GMRES solver: GMRES is handed the matrix-free matvec $v_{\text{real}} \mapsto \mathcal{E}(M^{-1}(A(\mathcal{E}^{-1}(v_{\text{real}}))))$ and the preconditioned, real-embedded right-hand side $\mathcal{E}(M^{-1}u^n)$, and returns an (approximate) solution once its residual falls below a prescribed tolerance or a maximum number of iterations is reached. Algorithm 5 summarizes the full procedure.

Since GMRES is an iterative method, the caller should always check whether it actually reached the requested tolerance within the allotted iterations, and decide how to handle the case where it did not (for example, by retrying with a smaller timestep or a larger iteration budget).

6.6 Locations in the Python code

Here are important locations in the code:

- `src/timestepping::make_implicit_op()` defines the action of the matrix-vector product $v \mapsto M^{-1}Av$.
- `src/timestepping::solve_step()` makes the call to GMRES.
- `src/real_embedding.py` implements utilities for $\mathcal{E}$ and $\mathcal{E}^{-1}$.

Algorithm 5: Solving one implicit backward Euler timestep via preconditioned GMRES

Input : Previous solution u^n (coefficients); timestep Δt ; inverse metric quantities V_θ, V_φ ; GMRES tolerance tol , restart/Krylov dimension r , maximum number of iterations maxit .

Output: Coefficients $\{(u^{n+1})_\ell^m\}$.

```

// Preconditioner eigenvalues, ??
1  $M_\ell \leftarrow 1 + \Delta t \ell(\ell + 1)$  for  $\ell = 0, \dots, L$ 
// Precondition the right-hand side, eq. (6.4)
2  $(M^{-1}u^n)_\ell^m \leftarrow (u^n)_\ell^m / M_\ell$ 
3 Zero the imaginary part of all  $\ell, m = 0$  coefficients of  $M^{-1}u^n$ 
4 Zero all  $\ell \geq L - 2$  coefficients of  $M^{-1}u^n$ 

// Map to the real embedding, eq. (6.5)
5  $b_{\text{real}} \leftarrow \mathcal{E}(M^{-1}u^n)$ 
6 define  $A_{\text{real}}(v_{\text{real}})$ :
7    $v \leftarrow \mathcal{E}^{-1}(v_{\text{real}})$ 
8   Compute  $\Delta_\Gamma v$  via Algorithm 4 using  $V_\theta, V_\varphi$ 
9    $w \leftarrow v + \Delta t \Delta_\Gamma v$ 
10   $(M^{-1}w)_\ell^m \leftarrow w_\ell^m / M_\ell$ 
11  Zero all  $\ell \geq L - 2$  coefficients of  $M^{-1}w$ 
12  return  $\mathcal{E}(M^{-1}w)$ 

// Restarted GMRES on the real, preconditioned system
13  $x_{\text{real}} \leftarrow \text{GMRES}(A_{\text{real}}, b_{\text{real}}; \text{tol}, r, \text{maxit})$ 

// Map back to coefficient space
14  $\{(u^{n+1})_\ell^m\} \leftarrow \mathcal{E}^{-1}(x_{\text{real}})$ 

```
