MQGeometry-1.0: a multi-layer quasi-geostrophic solver on non-rectangular geometries

Louis Thiry\textsuperscript{1}, Long Li\textsuperscript{1}, Guillaume Roullet\textsuperscript{2}, and Etienne Mémin\textsuperscript{1}

\textsuperscript{1}ODYSSEY, INRIA Rennes, IRMAR Rennes, Université de Rennes 1, France
\textsuperscript{2}Université de Bretagne Occidentale, CNRS, IRD, Ifremer, Laboratoire d’Océanographie Physique et Spatiale (LOPS), IUEM, Brest, France

Correspondence: Louis Thiry (louis.thiry@inria.fr)

Abstract. This paper presents MQGeometry, a multi-layer quasi-geostrophic (QG) equations solver for non-rectangular geometries. We advect the potential vorticity (PV) with finite volumes to ensure global PV conservation thanks to a staggered discretization of the PV and stream-function (SF). Thanks to this staggering, the PV is defined inside the domain, removing the need for defining the PV on the domain’s boundary. We compute PV fluxes with upwind-biased interpolations whose implicit dissipation replaces the usual explicit (hyper-)viscous dissipation. The presented discretization does not require the tuning of any additional parameter, e.g. additional eddy viscosity. We solve the QG elliptic equation with a fast discrete sine transform spectral solver on rectangular geometry. We extend this fast solver to non-rectangular geometries using the capacitance matrix method. We validate our solver on a vortex-shear instability test case in a circular domain, a vortex-wall interaction test-case, and on an idealized wind-driven double-gyre configuration in an octogonal domain at a eddy-permitting resolution. We release a concise, efficient, and auto-differentiable PyTorch implementation of our method to facilitate future developments upon this new discretization, e.g. machine learning parameterization or data-assimilation techniques.

1 Introduction

Ocean fluid dynamics offers a hierarchy of models that trade between richness of the physical phenomena and dimensionality of the system. On the one end of this hierarchy are the Boussinesq non-hydrostatic equations. These equations describe the evolution of the stratification via temperature and salinity, they model explicitly convective phenomena but they require the evolution of six prognostic variables: the three components of the velocity \( u, v, w \), temperature \( T \) and salinity \( s \), and the free surface \( \eta \).

On the other end of this hierarchy are the multi-layer quasi-geostrophic equations. These equations are based on strong hypotheses: static background stratification, hydrostatic and geostrophic balances. In a multi-layer QG model, the stream-function (SF) \( \psi \) and potential vorticity (PV) \( q \) are stacked in \( N \) isopycnal layers with density \( \rho_i \) and reference thickness \( H_i \):

\[
\psi = [\psi_1, \ldots, \psi_N]^T \quad \quad q = [q_1, \ldots, q_N]^T
\]
The governing equations read

\[ \partial_t q + \left( \begin{array}{c} u \\ v \end{array} \right) \cdot \nabla h q = 0, \quad \left( \begin{array}{c} u \\ v \end{array} \right) = \nabla_{\perp} h \psi, \] (1a)

\[ \Delta_h \psi - f_0^2 A \psi = q - \beta y, \] (1b)

\[ A = \begin{bmatrix} \frac{1}{H_{g1}'} + \frac{1}{H_{g2}'} & \frac{-1}{H_{g1}'} & \cdots & \frac{-1}{H_{g2}'} \\ \frac{-1}{H_{g1}'} & \frac{1}{H_{g2}'} + \frac{1}{H_{g2}'} & \cdots & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ \frac{-1}{H_{g2}'} & \frac{-1}{H_{g2}'} & \cdots & \frac{1}{H_{g2}'} \end{bmatrix} \] (1c)

\[ \nabla_{\perp} h = [-\partial_y, \partial_x]^T \] stands for the horizontal orthogonal-gradient, \( \Delta_h = \partial^2_{xx} + \partial^2_{yy} \) denotes the horizontal Laplacian. \( f_0 + \beta(y - y_0) \) is the Coriolis parameter under beta-plane approximation with the meridional axis center \( y_0 \). \( g_i' = g(\rho_{i+1} - \rho_i) / \rho_i \) are the reduced gravities.

QG equations hence involve a single prognostic variable, the potential vorticity (PV) \( q \), which is advected (eq. 1a). Despite strong hypotheses, the multi-layer QG equations are a robust approximation of ocean meso-scale non-linear dynamics. They offer a computationally efficient playground to study the meso-scale ocean dynamics and to develop physical parametrizations of the unresolved eddy dynamics (Marshall et al., 2012; Fox-Kemper et al., 2014; Zanna et al., 2017; Ryzhov et al., 2020; Uchida et al., 2022, e.g.).

Solving QG equations requires solving an elliptic equation (eq. 1b) which relates the streamfunction \( \psi \) and the potential vorticity \( q \). On a rectangular domain, it can be easily achieved using a fast spectral solver based on discrete Fourier transform (DFT) for periodic boundary conditions or discrete sine transform (DST) for no flow boundary conditions. Most of the available open-source QG solvers like Geophysical Flows (Constantinou et al., 2021), PyQG or Q-GCM (Hogg et al., 2014) use such spectral solvers and are therefore limited to rectangular geometries.

There are two major issues when implementing QG models on non-rectangular geometries. The first issue is the fact that spectral elliptic solvers do not apply for non-rectangular domains. In ocean models, one typically uses conjugate gradient (CG) iterative solvers like BiCGSTAB (Van der Vorst, 1992) to solve elliptic equations on non-rectangular domains, for example when solving the Poisson equations associated with a rigid lid-constraint (Häfner et al., 2021) or for implicit free-surface computations (Kevlahan and Lemarié, 2022) in Primitive equation solvers. These CG iterative solvers are significantly slower than spectral solvers to solve Poisson or Helmholtz equations on evenly spaced rectangular grids (Brown, 2020). Using them in a QG solver would reduce significantly the computational efficiency that makes QG appealing compared to Shallow-water (SW) models.

The second issue is the definition of the potential vorticity (PV) on the boundaries. If we discretize the potential vorticity \( q \) on the same locations as the streamfunction \( \psi \), we have to define the potential vorticity on the boundaries. This requires using partial free-slip/no-slip condition (Hogg et al., 2014, see e.g.) to define ghost points in order to compute the laplacian of the streamfunction. This ad-hoc definition might violate the global conservation of PV.
In this paper, we present MQGeometry, a new multi-layer QG equations solver that addresses these two issues. This solver uses a new discretization of the multi-layer QG equations based on two main decisions. The first decision is to discretize the potential vorticity (PV) and the stream-function (SF) on two staggered grids. When doing so, the PV is not defined on the domain’s boundary but in its interior. This solves the issue of defining the PV on the domain boundaries. Moreover, this decision allows using a finite-volume scheme for the PV advection. It warrants the global conservation of the PV and leverages a fine control on the PV fluxes. The second decision is to solve the elliptic equation using a fast spectral DST solver combined with the capacitance matrix method (Proskurowski and Widlund, 1976) to handle non-rectangular geometries.

In addition to these two decisions, we decide to compute PV fluxes with upwind-biased stencil interpolation to remove the additional (hyper-)viscosity used in most discrete QG models. Doing so, we explore a different approach that is complementary to physical parameterizations of the horizontal momentum closure: a careful choice of the numerical schemes used to discretize the continuous equations. This idea has been initially developed by Boris et al. (1992) in the context of large eddy simulations (LES) models and made popular as implicit LES by Grinstein et al. (2007). It has been successfully tested by Von Hardenberg et al. (2000) to study vortex merging in a single layer QG model and by Roullet et al. (2012) to study the forced-dissipated three-dimensional QG turbulence in a channel configuration. While Von Hardenberg et al. (2000) and Roullet et al. (2012) used only linear upwind-biased reconstruction, we implement here linear and non-linear weighted essentially non oscillatory (WENO) reconstructions (Jiang and Shu, 1996; Borges et al., 2008) that are tailored to remove spurious numerical oscillations created by linear reconstructions (Liu et al., 1994).

We implement the proposed discretizations in a concise Python-PyTorch code that allows seamless GPU acceleration. It benefits from built-in automatic differentiation to facilitate future development in machine-learning or data-assimilation. We validate our solver on a vortex-shear instability test case in a closed squared domain, an inviscid vortex-wall interaction, and an idealized wind-driven double-gyre configuration in a non-rectangular configuration.

This paper is organized as follows. In Section 2, we describe the resolution of the PV advection equation with finite volume. In Section 3, we present our elliptic solver based on discrete sine transform and capacitance matrix method (Proskurowski and Widlund, 1976). In Section 4, we detail the solver implementation. In Section 5, we describe the experimental settings to validate our discretization. We conclude and evoke further perspectives in Section 6.

2 PV advection with finite volumes

2.1 Staggered discretization of PV and SF

The first ingredient of our method is to use finite-volumes to solve PV advection (eq. 1a). This leads us naturally to a staggered discretization for the PV and the SF (see right panel of Figure 1). With this choice, the PV advection (1a) can be integrated over the whole domain as a transported tracer with a finite volume formulation. Indeed, if \( \psi \) is discretized at the cell vertices,
the orthogonal-gradient of $\psi$ computed with the standard second order discretization

$$ \nabla^\perp_{h} f_{i,j} = \left( \begin{array}{c} \frac{f_{i,j+1} - f_{i,j}}{\delta y} \\ \frac{f_{i+1,j} - f_{i,j}}{\delta x} \end{array} \right) $$

lives in the middle of the cell edges. The horizontal velocity $u$ lives in the middle of the vertical egdes while the vertical velocity $v$ lives in the middle of the vertical edges. We thus have to discretize the PV $q$ at the cell centers to solve its advection with finite volumes.

At the boundary which passes along the cell edges, the condition is simply the no-flux of PV across the walls. This staggering clearly separates the boundary condition associated with the transport of PV from the boundary condition associated with friction. Moreover, the global conservation of PV during advection is ensured up to numerical precision with finite volumes.

In the usual discretization (see Appendix A), the PV must be defined on the boundary, which requires defining the Laplacian operator $\Delta_{h}$ there. This is problematic since it forces blending the slip boundary conditions into the definition of the Laplacian, rather than having a slip boundary condition clearly separated from the definition of PV. Moreover, depending on this choice, the material conservation of the PV might not be ensured even though the advection scheme, e.g. Arakawa-Lamb (Arakawa and Lamb, 1981), conserves the PV inside the domain.
Figure 2. Illustration of the upwind biased reconstruction. Away from boundaries we use a 5-points stencil for reconstruction, while close to boundaries, we use a 3-points stencil when possible (top-left). At distance one from boundary when the velocity goes away from the boundary (bottom-right), we use a second order reconstruction with a two-points centered stencil rather than upwind-1 reconstruction to have a reconstruction of order at least two. For linear reconstructions, the weights $c_i$ are fixed. For non-linear reconstruction, the weights $c_i$ depend on the solution.

2.2 Upwinding of PV fluxes

In a finite volume formulation, we rewrite the PV advection (1a) as the divergence of a flux:

$$\partial_t q = -\nabla_h \cdot \left( \begin{pmatrix} u \\ v \end{pmatrix} q \right).$$

(2)

The divergence is discretized with the usual second-order finite differences operator

$$\nabla_h \cdot \begin{pmatrix} u_{i,j} \\ v_{i,j} \end{pmatrix} = \frac{u_{i+1,j} - u_{i,j}}{\delta x} + \frac{v_{i,j+1} - v_{i,j}}{\delta y}.$$

We hence need to interpolate $q$ on the $u$ and $v$ grid points to compute these PV fluxes. In the context of finite volume methods, we call these interpolations reconstructions.

For a good trade-off between stability and accuracy, we use a 5-points upwind-biased stencil for reconstruction. Near the boundary we use a 3-points upwind-biased stencil or a 2-points centered stencil as illustrated in Figure 2. The ordering of
Figure 3. One dimensional advection with constant velocity on a periodic domain over one period. We compare 3 finite volume methods using linear, WENO-JS (Jiang and Shu, 1996), and WENO-Z (Borges et al., 2008) 5-points upwind reconstructions.

The stencil is given by the sign of the velocity. For instance, \( q_i = \sum_{s=1}^{5} c_s q_s \) when the velocity is positive, and we have the reverse order when the velocity is negative. Using upwind biased stencils for reconstruction allows removing additional ad-hoc (hyper-)viscosity which is necessary when using centered reconstructions (Lemarié et al., 2015). We decide here to rely solely on the upwinding to handle the potential enstrophy dissipation, which is done implicitly.

We offer two possibilities for reconstructions: linear reconstructions or Weighted Essentially Non Oscillatory (WENO) reconstructions (Liu et al., 1994). The weights \( c_s \) are fixed for the linear reconstruction. However, flux computed with linear reconstructions tend to produce spurious numerical oscillations when the field \( q \) is not smooth (see Figure 3).

WENO reconstructions benefit from the essentially non oscillatory property (Harten, 1984): they are designed to prevent spurious numerical oscillations that occur with linear reconstructions (see Figure 3). With a WENO scheme, the weights \( c_s \) are not fixed: they depend on the value of \( q \) on the five-point stencil via smoothness indicators. The smoother \( q \) on the stencil, the closer the weights \( c_s \) are to the linear weights. One can find a detailed explanation of these non-linear weights computations in Borges et al. (2008).
2.3 Time integration

WENO schemes’ convergence and stability properties derived in Jiang and Shu (1996) require using TVD Runge-Kutta scheme of order at least 3 (Shu and Osher, 1988). Here we use Runge-Kutta TVD scheme of order 3 for time integration with finite time step $\delta t$.

Given the time ordinary differential equation

$$\partial_t f = Lf,$$

one can write the RK3 TVD scheme with the following three stages:

$$f^{(1)} = f^{(0)} + \delta t Lf^{(0)}$$

$$f^{(2)} = f^{(1)} + \frac{\delta t}{4} \left( Lf^{(1)} - 3Lf^{(0)} \right)$$

$$f^{(3)} = f^{(2)} + \frac{\delta t}{12} \left( 8Lf^{(2)} - Lf^{(1)} - Lf^{(0)} \right).$$

This low memory version requires only the storage of the intermediate time derivatives $Lf^{(i)}$.

3 Spectral elliptic solver on non-rectangular domain

3.1 Staggering of PV and SF

After having solved PV advection equation (Eq. (1a)), one has to solve the elliptic equation (1b) to compute the SF. The SF $\psi$ satisfies the homogeneous Dirichlet boundary condition, hence we have to compute the r.h.s. terms inside the domain. Due to the staggered discretization, we need to interpolate them between the two grids. A natural way to proceed is to use a 4-points linear interpolation to interpolate $q$ on the $\psi$-grid (see Figure 4).

3.2 DST solver on rectangular geometry

To solve the elliptic equation (1b), we use a vertical transform and a fast spectral solver with discrete sine transform (DST). One can diagonalize the matrix $A$ (defined in eq. 1c) as follows

$$A = C_{m2l} \Lambda C_{l2m},$$

where the layer-to-mode matrix $C_{l2m}$ is the inverse of the mode-to-layer matrix $C_{m2l}$, and $\Lambda$ is a diagonal matrix containing $A$ eigenvalues. One can then perform the following layer-to-mode transform:

$$\tilde{\psi}, \tilde{q} = C_{l2m} \psi, C_{l2m} q.$$

With this transform, the elliptic equation (1b) becomes a stack of $N$ two-dimensional Helmholtz equations

$$\Delta_h \tilde{\psi} - f_0^2 \Lambda \tilde{\psi} = \tilde{q} - \beta y$$

(3)
with homogeneous Dirichlet boundary conditions. To solve these \( N \) two-dimensional Helmholtz equations, we use fast-diagonalization with type-I discrete sine transform (DST-I) since the usual 5-points Laplacian operator
\[
\Delta_h f_{i,j} = \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{\delta x^2} + \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j+1}}{\delta y^2}
\]
becomes a diagonal operator in the type-I sine basis (Press and Teukolsky, 2007, 20.4, p. 1055). After having solved these \( N \) Helmholtz equation, we transform back \( \psi \) from mode to layers
\[
\psi = C_{m2\ell} \tilde{\psi}.
\]

### 3.3 Capacitance matrix method for non-rectangular geometry

To handle non-rectangular domains, we use the capacitance matrix method (Proskurowski and Widlund, 1976). We have a non-rectangular domain \( \Omega = \partial \Omega \cup \tilde{\Omega} \) where \( \partial \Omega \) is the domain boundary and \( \tilde{\Omega} \) is the interior of the domain. Our non-rectangular domain \( \Omega \) is embedded in a rectangular domain \( \Omega_R = \partial \Omega_R \cup \tilde{\Omega}_R \). We denote by \( \mathcal{I} = \partial \Omega \setminus \partial \Omega_R \) the set of non-rectangular boundary indices (see Figure 5). We assume that we have \( K \) non-rectangular boundary points \( I_k \in \mathcal{I} \).

We now explain how to solve the following non-rectangular Helmholtz equation
\[
\Delta_h f - \lambda f = r \text{ in } \tilde{\Omega}, \quad \lambda \in \mathbb{R}^+, \quad f = 0 \text{ on } \partial \Omega.
\] (4)

One can find a more detailed explanation in Blayo and LeProvost (1993) as this is our inspiration for using the capacitance matrix method.
Figure 5. Domain $\Omega$ included in a rectangular domain $\Omega_R$, and the set $I = \partial \Omega \setminus \partial \Omega_R$ of non-rectangular boundary indices.

### 3.3.1 Pre-computations

For each non-rectangular boundary point $I_k \in I$, a Green function $g_k$ is defined as the solution of the following rectangular Helmholtz equation:

$$\Delta_h g_k - \lambda g_k = \begin{cases} 
1 & \text{on } I_k \\
0 & \text{in } \Omega_R \setminus \{I_k\}
\end{cases}$$

$$g_k = 0 \text{ on } \partial \Omega_R .$$

solved using DST-I fast-diagonalization. With these Green functions, we compute the square matrix $M$ with coefficients

$$m_{k,l} = g_l(I_k) .$$

We compute this matrix inverse to get the so-called capacitance matrix $C = M^{-1}$. 
3.3.2 First-step

In a first step, we solve the following rectangular Helmholtz equation

$$\Delta_h f^{(1)} - \lambda f^{(1)} = \begin{cases} \text{in } \hat{\Omega} & r \\ 0 \text{ in } \hat{\Omega}_R \setminus \hat{\Omega} & 0 \end{cases}$$

$$f^{(1)} = 0 \text{ on } \partial \Omega$$

using DST-I fast-diagonalization. We then compute the vector $s$ coefficients

$$s_k = f^{(1)}(I_k)$$

and we deduce the vector

$$\alpha = -Cs$$

3.3.3 Second-step

In a second step, we solve the following rectangular Helmholtz equation

$$\Delta_h f^{(2)} - \lambda f^{(2)} = \begin{cases} \text{in } \hat{\Omega} & r \\ \alpha_k \text{ on } I_k & \alpha_k \\ 0 \text{ in } \hat{\Omega}_R \setminus \hat{\Omega} & 0 \end{cases}$$

$$f^{(2)} = 0 \text{ on } \partial \Omega_R$$

using DST-I fast-diagonalization. This function $f^{(2)}$ is such that

$$\Delta_h f^{(2)} - \lambda f^{(2)} = r \text{ in } \hat{\Omega}$$

$$f^{(2)}(I_k) = 0 \forall I_k \in \mathcal{I}$$

$$f^{(2)} = 0 \text{ on } \partial \Omega \setminus \mathcal{I}$$

The restriction of $f^{(2)}$ over our non-rectangular domain $\Omega$ is therefore solution of the Helmholtz equation (4).

4 Implementation

4.1 Programming language and library

One can implement the above discretization using any programming language. To anticipate later applications in data assimilation and machine learning, and to benefit easily from GPU acceleration, we have decided to use the PyTorch library (Paszke et al., 2019). This library contains the usual linear algebra routines, and the computations can be easily vectorized. It offers a built-in automatic differentiation to compute gradients or adjoints, a feature that we do not use in the present study.
4.2 Upwind flux computation

Upwind flux computations require splitting the velocity into positive and negative part. This can be achieved very efficiently using the ReLU function

\[
\text{ReLU}(x) = \max(x, 0)
\]

whose PyTorch implementation is highly optimized since this function is widely used in neural networks. The positive part \( u^+ \) and negative part \( u^- \) of the velocity are given by

\[
\begin{align*}
    u^+ &= \text{ReLU}(u) \\
    u^- &= u - u^+
\end{align*}
\]

4.3 WENO reconstructions

Weighted essentially non-oscillatory (Liu et al., 1994, WENO) is a large class of reconstruction methods. We implement two of the most widely used WENO reconstructions: the WENO-JS (Jiang and Shu, 1996) and the WENO-Z (Borges et al., 2008). Implementing these methods require only a few lines of Python code (written in Appendix B).

4.4 Masks

For non-rectangular domain, one has to provide a binary mask for the PV grid with ones inside the domain and zeros outside. Given this mask, we automatically compute the masks for the other variable, i.e. the stream-function \( \psi \) and the two components of the velocity \( u \) and \( v \), and we deduce the non-rectangular boundary set \( I \) involved in the capacitance matrix computations. We also compute specific masks that specify the stencil for PV reconstruction on \( u \) and \( v \) points for PV fluxes computations. This is illustrated in Figure 6.

4.5 Elliptic solver

Our elliptic solver is based on fast diagonalization using discrete sine transform. Since PyTorch implements FFT but not DST, we implement DST-I using FFTs with specific pre- and post-processing operations. Notably, our DST-I implementation is faster than Scipy’s implementation on Intel CPUs thanks to PyTorch bindings to MKL FFT.

The capacitance matrix method implementation is straightforward following the equations in section 3. Capacitance matrices are precomputed given the set of non-rectangular boundary indices \( I \). The method solves the Helmholtz equation with machine precision accuracy. Figure 7 illustrates this point on a circular domain at resolution \( 256^2 \). From a prescribed streamfunction \( f \), taken as Gaussian white noise and vanishing along the boundary, we apply the Helmholtz operator to get a r.h.s. We then solve the Helmholtz equation with this r.h.s. to get \( f_{\text{inv}} \). Figure 7 shows that \( f_{\text{inv}} - f \) is of the order of machine precision. The method is independent of the domain shape. The numerical experiments below explore various domain shapes.
Figure 6. Illustration of the different masks. The label “u ic2” means that one uses two-points centered stencils to reconstruct $q$ on these $u$ points for PV fluxes computations. The label “v iup5” means that one uses five-points upwind-biased stencils to reconstruct $q$ on these $v$ points for vertical PV fluxes computations.

4.6 Compilation

PyTorch embeds a compiler which allows a significant speed-up for computationally intense routines. We compile the flux computations and finite difference routines. We get a 2.2× speed-up thanks to this compilation with `torch.compile`.

4.7 Ensemble simulations

Thanks to PyTorch vectorized operations, our implementation allows running ensemble simulations, i.e. parallel simulations starting from different initial conditions. This is a promising possibility for later developments of ensemble-based data assimilation or for stochastic extensions such as (Li et al., 2023).
Figure 7. Example of Helmholtz equation solved with our solver on a circular domain embedded in a $256^2$ square domain. The true function $f$ is initialized with spatially uncorrelated Gaussian white noise. The inverted function $f_{inv}$ is computed using our solver. The difference between $f$ and $f_{inv}$ is of the order of machine precision.

4.8 Architecture

The code is divided into six python scripts:

- `helmholtz.py` which contains the Helmholtz solvers based on DST-I and capacitance matrix method.
- `fd.py` which contains the finite difference functions.
- `reconstruction.py` which contains the reconstructions (i.e. interpolations in the context of finite volumes) routines.
- `masks.py` which contains the mask utility module.
- `flux.py` which contains the flux computations routines.
- `qgm.py` which contains the python class implementing the multi-layer QG model.

We end up with a concise PyTorch implementation ($\sim 750$ lines of code) which is as close as possible to the equations. It can run seamlessly on CPUs and GPUs with CUDA compatibility.
4.9 Performance

On a Dell precision 7560 Laptop, we get an execution time of $4.1 \times 10^{-9}$ sec. per grid-point per RK step on GPU (NVIDIA GeForce RTX 2080 Ti), and $4.8 \times 10^{-8}$ sec. per grid-point per RK step on a single core CPU (Intel Core i9-11950H).

![Evolution of potential vorticity](image)

**Figure 8.** Vortex-shear instability experiment. Evolution of potential vorticity $q$ at initial time $t = 0$, at intermediate times $5\tau, 7.5\tau, 9\tau, 13\tau, 18\tau, 24\tau$, and at final simulation time $30\tau$.

5 Numerical validation

We run three numerical experiments involving meso-scale vortices (20 to 200 km diameters) to validate our solver. The first one is a vortex shear instability, the second one is a vortex wall interaction, and the third one is an idealized double-gyre experiment, which is a usual toy model for western boundary currents. These experiments With our solver, multi-layer QG equations deliver on their promise of computationally efficient playground to study meso-scale non-linear dynamics. Indeed, the three presented experiments ran on a laptop and took a few to to fifty minutes to run. We provide the python scripts to reproduce these experiments and the figures.
5.1 Vortex shear instability

The first validation of our method consists of a vortex shear instability. We consider a rotating fluid in a circular domain with diameter $D = 100$ km embedded in a square domain of size $L_x \times L_y = 100$ km $\times$ 100 km on a f-plane with a Coriolis parameter $f_0$, whose value is deduced below from the Burger number below. There is a single layer of fluid with reference thickness $H = 1$ km. We assume no-flow and free-slip boundary conditions. The gravity constant is set to $g = 10$ m s$^{-2}$.

We study the shear instability of a meso-scale shielded Rankine vortex, which has piece-wise constant potential vorticity. In the initial state, the vortex is composed of a core vortex surrounded by a ring of opposite-sign potential vorticity to the core such that the total sum of the PV is zero. This system is shear unstable and generates multipoles (Morel and Carton, 1994). We focus here on the tripole formation regime.

The core of the vortex has a radius $r_0 = 10$ km and positive vorticity. The surrounding ring has an inner radius $r_1 = 14$ km. The remaining parameters of the simulation are set via the Rossby and Burger numbers defined as follows

$$Ro = \frac{u_{\text{max}}}{f_0 r_0},$$

$$Bu = \frac{gH}{(f_0 r_0)^2},$$

where $u_{\text{max}}$ is the maximum velocity of the initial condition. Given the Burger number, we compute the Coriolis parameter using $f_0 = \sqrt{\frac{gH}{Bu r_0^2}}$. Then given the Rossby number, we rescale the velocity field of the initial condition such that the maximum velocity is $u_{\text{max}} = Ro f_0 r_0$.

The quasi-geostrophic equations are valid for $Bu \leq 1$ and $Ro \ll 1$. We perform the experiment with $Ro = 0.01$ and $Bu = 1$. The experiment is integrated over a period of $30\tau$, with $\tau = \|q_{\text{init}}\|_2^{-1}$ the eddy-turnover time, and $q_{\text{init}}$ the initial condition, shown at the top-left of Figure 8. At initial time the PV contours of the core and the ring $r = r_1$ are slightly perturbed by means of a mode three azimuthal perturbation defined in polar coordinates $(r, \theta)$ by

$$(1 + \epsilon \cos(3\theta)) r = r_1,$$

where $\epsilon \ll 1$ is a small parameter, typically $\epsilon = 0.001$. This perturbation favors the growth of the most unstable mode which, given the ratio $r_1/r_0$, evolves non-linearly into a tripole (Morel and Carton, 1994).

We use WENO-Z reconstructions for this experiment. This experiment can be reproduced with script `vortex_shear.py`. We run the reference experiment at a resolution of $1024^2$.

We see on Figure 8 the evolution of the vortex potential vorticity $q$ at different times between the beginning and the end of the simulation at time $t = 30\tau$. At time $t = 5\tau$ the result of the instability is visible and the core vortex which was initially circular has a triangular shape. At time $t = 7.5\tau$ the outer positive PV ring has become the expected tripole (Morel and Carton, 1994). At time $t = 9\tau$ the core vortex recovers a triangular shape with negative PV filaments ranging from the triangle vertices to the three positive vortices. At time $t = 13\tau$, the core vortex keeps a triangular shape and the vorticity filaments are becoming thinner. At time $t = 18\tau$, the filaments thickness are reaching the grid scale and they start being dissipated by WENO-Z implicit dissipation. At time $t = 24\tau$, the filaments have are being progressively dissipated and have a smaller amplitude. The
core vortex shape is become more circular, and the three positive vortices are surrounded by a smaller negative vortex. At final time $t = 30\tau$, the filaments amplitude have lowered, and small vorticity dipoles appear. The order three symmetry, that was injected by the initial perturbation, starts to be lost only at time $t = 30\tau$. For later times (not shown), the system evolves into a chaotic system. This chaotic evolution is expected as the system is sensitive to initial conditions.

To measure the sensitivity of our solver to resolution, we consider the $1024^2$ resolution as the high resolution reference, and we run the simulation at two lower resolutions: $256^2$, and $512^2$. At the top of Figure 9 we plot the evolution of the total enstrophy at the three different resolutions. The results show that the enstrophy dissipation decreases as the resolution increases, indicating that a higher resolution better preserves the PV variance, and that low resolution simulations suffer from an excessive dissipation.

5.2 Vortex-wall interaction

To challenge the numerics we now study the propagation of a single meso-scale vortex along a solid boundary, with a free slip boundary condition. The domain is square with a thin wall obstacle. Because the flow is inviscid, up to numerical errors, the vortex is expected to follow the boundary according to the mirror effect. In particular the vortex should slip around the obstacle and litteraly circumvent it, without detaching from it, without producing filaments (Deremble et al., 2016). The challenge is thus to have a solution as inviscid as possible, and to enforce as better as possible both the no-flow and the free-slip boundary conditions.
Figure 10. Vortex-wall interaction experiment. Vortex potential vorticity $q$ and streamfunction $\psi$ contours at initial time $t = 0$, at intermediate times $3.1\tau, 6.3\tau, 9.4\tau, 12.6\tau, 15.7\tau, 18.9\tau$, and at final simulation time $22\tau$.

The setup is as follows. The domain size is $L_x \times L_y = 100\text{ km} \times 100\text{ km}$ on a f-plane with a Coriolis parameter $f_0$. The thin wall obstacle is vertical starting from the middle of the domain south boundary and of length $L_y/4$ (see Figure 10) and has a two cells width. There is a single layer of fluid with reference thickness $H = 1\text{ km}$. We assume no-flow and free-slip boundary conditions. The gravity constant is set to $g = 10\text{ m s}^{-2}$. In the initial state, the vortex is a circle of radius $r_0 = 10\text{ km}$ with constant potential vorticity and is at the bottom of the domain and at the left of the wall. The circular shape differs from the oval shape a vortex has when moving along a rectilinear wall. The remaining parameters of the simulation are set via the Rossby and Burger with $Ro = 0.01$, and $Bu = 1$.

Figure 10 shows PV snapshots at various times, superimposed with the streamfunction. The vortex behaves according to the inviscid regime. It clearly follows the wall and the obstacle elastically, without any sign of dissipative process such as filament detachment. The vortex at $t = 22\tau$ has recovered the characteristic oval shape. Between $t = 9.4\tau$ and $t = 12.6\tau$ the vortex circumvents the edge which causes it to experience its maximal deformation, but the solution remains very smooth. The streamfunction is clearly constant along the boundary, as a direct consequence of the capacitance matrix method. During the circumvention, it remains so, despite the thin wall imposing a strong curvature at the edge.

This experiment shows that the numerics has very good conservation properties on inviscid flows. This may come as a surprise since the upwinding does induce a numerical dissipation. In practice, the dissipation seems to self adjust to the minimum
Table 1. Parameters of the idealized double-gyre configuration

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_x \times L_y$</td>
<td>$(5120 \times 5120)$ km</td>
<td>Domain size</td>
</tr>
<tr>
<td>$n_x \times n_y$</td>
<td>$256 \times 256$</td>
<td>Grid dimension</td>
</tr>
<tr>
<td>$dx \times dy$</td>
<td>$20 \times 20$ km</td>
<td>Spatial resolution</td>
</tr>
<tr>
<td>$H_k$</td>
<td>$(400, 1100, 2600)$ m</td>
<td>Layer thickness</td>
</tr>
<tr>
<td>$g'_k$</td>
<td>$(0.025, 0.0125)$ m s$^{-2}$</td>
<td>Reduced gravity</td>
</tr>
<tr>
<td>$\delta$</td>
<td>$1.43 \times 10^{-5}$ s$^{-1}$</td>
<td>Bottom drag coeff.</td>
</tr>
<tr>
<td>$\tau_0$</td>
<td>$0.08$ N m$^{-2}$</td>
<td>Wind stress magnitude</td>
</tr>
<tr>
<td>$\rho_0$</td>
<td>$1000$ kg m$^{-3}$</td>
<td>Ocean density</td>
</tr>
<tr>
<td>$f_0$</td>
<td>$9.375 \times 10^{-5}$ s$^{-1}$</td>
<td>Mean Coriolis</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$1.754 \times 10^{-11}$ (m s)$^{-1}$</td>
<td>Coriolis gradient</td>
</tr>
<tr>
<td>$L_d$</td>
<td>$(41, 25)$ km</td>
<td>Rossby radii</td>
</tr>
<tr>
<td>$dt$</td>
<td>$4000$ s</td>
<td>Time-step</td>
</tr>
</tbody>
</table>

required to prevent noise at the grid scale. This way of discretizing, in line with the ILES approach, turns out to be a viable alternative to conservative discretization combined with an explicit dissipation term.

5.3 Double-gyre configuration

Our third numerical experiment to validate our solver is an idealized double-gyre configuration. Double-gyre configuration are a natural test for QG implementation or parameterization (Zanna et al., 2017; Ryzhov et al., 2020; Uchida et al., 2022). We consider here an octogonal ocean basin to illustrate the ability of our solver to handle non-square geometries. This octogon has maximal dimensions $L_x \times L_y$. We assume free-slip boundary conditions on the boundaries. We consider $N = 3$ layers on the vertical. We use an idealized stationary and symmetric wind stress $(\tau_x, \tau_y)$ with $\tau_x = -\left(\tau_0/\rho_0\right)\cos(2\pi y/L_y)$ and $\tau_y = 0$ on the top and linear drag at the bottom drag coefficient $\delta$. The parameter values are given in Table 1.

We study this configuration in an eddy-permitting resolution of 20km the eddy-resolving meaning that the spatial resolution (20km) is half of the larger baroclinic Rossby radius (41km). Already at such eddy-permitting resolution, multi-layer QG solvers do not necessarily produce a well-pronounced eastward jet and usually require additional eddy parameterization (Uchida et al., 2022).

We plot on top of Figure 11 a snapshot of upper-layer streamfunction and relative vorticity (i.e. $\Delta \psi$) after 30 years of spin-up. As expected, our solver produces a strong western boundary current on the vertical and non-vertical boundaries. In the middle of this boundary starts a well extended eastward jet whose length is qualitatively comparable with the Gulf-stream length in the North-Atlantinc basin. This jet is surrounded by a recirculation zone with several meso-scale eddies, which appear
coherent with eddy-resolving resolution simulations. We notice large scale Rossby waves that are emerging near the eastern boundary and that propagate westward.

Since this system is chaotic, showing a snapshot is not very representative of the dynamical behaviour of the system. We plot statistics of our solution on Bottom of Figure 11: the mean stream-function, the mean kinetic energy (i.e. the kinetic energy of the mean velocity), and the eddy kinetic energy (i.e. the kinetic energy of the velocity standard deviation). These statistics were computed over 40 years after 10 years spin-up, saving one snapshot every 15 years. These statistics are symmetric which is expected since the domain shape and the wind forcing are symmetric. They confirm the presence of a strong western boundary currents and fluctuating eastward jet whose length is roughly three fourth of the domain. These results seem to confirm the relevance of implicit dissipation provided by upwinding, since usual multi-layer QG solvers require additional eddy parameterization (Uchida et al., 2022) to produce a well-pronounced eastward jet.

Figure 11. (Top) Upper-layer stream function $\psi$ and relative vorticity (i.e. $\Delta \psi$) after 50 years of spin-up. (Bottom) Upper-layer mean stream function, mean kinetic energy density, and eddy kinetic energy density computed over 40 years after 10 years of spin-up.
6 Conclusions

We presented MQGeometry, a multi-layer quasi-geostrophic equations solver for non-rectangular geometries. This solver has three original aspects compared to usual solvers like Q-GCM, PyQG or PEQUOD: the use of finite-volume for advection via staggering of the potential vorticity and stream-function, the non-linear WENO upwind-biased reconstructions with implicit dissipation, and the ability to handle non-square geometry with a fast spectral DST solver combined with the capacitance matrix method. Running a simulation with this solver does not require the tuning of any additional parameter, e.g. additional hyper-viscosity.

This multi-layer QG solver delivers a computationally efficient playground to study meso-scale non-linear dynamics. It opens the path to study the QG dynamics in basin with realistic coast line, e.g. Mediterranean or North-Atlantic basins. Moreover, with PyTorch automatic differentiation, one can easily build upon this implementation to develop new machine learning parameterizations of the QG sub-grid scales or new data-assimilation techniques using QG.

Code availability. The python source code to reproduce the results is attached in the supplementary materials for reviewers. It contains a readme file with the instructions to run the code and a script to compute statistics and reproduce the figures. It will be available online at publication time.

Appendix A: Usual discretization of multi-layer QG equations.

One typically solves multi-layer QG equations using the following strategy (Hogg et al., 2014, e.g.) :

1. Use an evenly spaced Arakawa C-grid with the PV and the SF discretized on the same location, namely the cell vertices (see left panel of Figure 1).

2. Solve the PV advection equation (1a) using energy-enstrophy conservative Arakawa-Lamb scheme (Arakawa and Lamb, 1981) in the interior domain (i.e. not on the boundaries).

3. Since the scheme is energy conserving, use an additional (hyper-)viscosity scheme to dissipate the energy fueled by the wind forcing.

4. Given the matrix $A$ diagonalization

   \[ A = C_{m2l} \Lambda C_{l2m}, \]

   where the layer-to-mode matrix $C_{l2m}$ is the inverse of the mode-to-layer matrix $C_{m2l}$, and $\Lambda$ is a diagonal matrix containing $A$ eigenvalues, perform the following layer-to-mode transform:

   \[ \tilde{\psi}, \tilde{q} = C_{l2m} \psi, C_{l2m} q. \]
With this transform, the elliptic equation (1b) becomes a stack of $N$ two-dimensional Helmholtz equations

$$\Delta_h \tilde{\psi} - f_0^2 \lambda \tilde{\psi} = \tilde{q} - \beta y$$  \hspace{1cm} (A1)

with homogeneous Dirichlet boundary conditions.

5. Solve these $N$ two-dimensional Helmholtz equations, using e.g. fast-diagonalization with type-I discrete sine transform (DST-I):

$$\text{DST-I}[x]_k = \sum_{l=1}^{L} x_l \sin \left( \frac{\pi l k}{L+1} \right), \quad k = 1 \ldots L.$$  

6. Transform back from mode to layers

$$\psi, \ q = C_{m2l} \tilde{\psi}, \ C_{m2l} \tilde{q}.$$  

7. Update the PV boundary values using the elliptic equation (1b). This requires defining the Laplacian on the boundaries and possibly involves partial free-slip/no-slip boundary condition.

---

**Appendix B: WENO implementation**

```python
1: def weno5(qmm, qm, q0, qp, qpp):
2:     """ WENO Jiang Shu 1996."""
3:     eps = 1e-8
4:     q1 = 1/3*qmm - 7/6*qm + 11/6*q0
5:     q2 = -1/6*qm + 5/6*q0 + 1/3*qp
6:     q3 = 1/3*q0 + 5/6*qp - 1/6*qpp
7:     k1, k2 = 13/12, 0.25
8:     beta1 = k1 * (qmm-2*qm+q0)**2
9:         + k2 * (qmm-4*qm+3*q0)**2
10:    beta2 = k1 * (qm-2*q0+qp)**2
11:        + k2 * (qm-qp)**2
12:    beta3 = k1 * (q0-2*qp+qpp)**2
13:        + k2 * (3*q0-4*qp+qpp)**2
14:     g1, g2, g3 = 0.1, 0.6, 0.3
15:     w1 = g1 / (beta1+eps)**2
16:     w2 = g2 / (beta2+eps)**2
17:     w3 = g3 / (beta3+eps)**2
```

https://doi.org/10.5194/egusphere-2023-1715

Preprint. Discussion started: 27 September 2023

© Author(s) 2023. CC BY 4.0 License.
20:
21:     qi_weno5 = (w1*qi1+w2*qi2+w3*qi3) /
22:            (w1+w2+w3)
23:
24:     return qi_weno5
25:
26: def weno5z(qmm, qm, q0, qp, qpp):
27:     """ WENO-Z Borges et al. 2008"""
28:     eps = 1e-14
29:     qi1 = 1/3*qmm - 7/6*qm + 11/6*q0
30:     qi2 = -1/6*qm + 5/6*q0 + 1/3*qp
31:     qi3 = 1/3*q0 + 5/6*qp - 1/6*qpp
32:
33:     k1, k2 = 13/12, 0.25
34:     beta1 = k1 * (qmm-2*qm+q0)**2
35:        + k2 * (qmm-4*qm+3*q0)**2
36:     beta2 = k1 * (qm-2*q0+qp)**2
37:        + k2 * (qm-qp)**2
38:     beta3 = k1 * (q0-2*qp+qpp)**2
39:        + k2 * (3*q0-4*qp+qpp)**2
40:
41:     tau = torch.abs(beta1 - beta3)
42:     g1, g2, g3 = 0.1, 0.6, 0.3
43:     w1 = g1 * (1 + tau/(beta1+eps))
44:     w2 = g2 * (1 + tau/(beta2+eps))
45:     w3 = g3 * (1 + tau/(beta3+eps))
46:
47:     qi_weno5 = (w1*qi1+w2*qi2+w3*qi3) /
48:            (w1+w2+w3)
49:
50:     return qi_weno5

Author contributions. L.T. implemented the Pytorch software, conducted the numerical experiments, and wrote the article. G.R. gave the original idea of the staggering and the use of WENO, supervised the numerical experiments, and corrected the paper. L.L. helped with numerical experiments. E.M. supervised the project.
Competing interests. The authors declare no competing interests.

Acknowledgements. The authors acknowledge the support of the ERC EU project 856408-STUOD. We would like to express our warm thanks to Laurent Debreu for pointing us to the capacitance matrix method. We also thank Nicolas Crouseilles, Pierre Navaro, Erwan Faou, and Georges-Henri Cottet for the amicable exchanges and interesting discussions.
References


