The Journal of Open Source Software

xinvert: A Python package for inversion problems in geophysical fluid dynamics

Yu-Kun Qian [1¶]

1 State Key Laboratory of Tropical Oceanography, South China Sea Institute of Oceanology, Chinese Academy of Sciences, Guangzhou, China ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Statement of need

Many problems in meteorology and oceanography can be cast into a balanced model in the form of a partial differential equation (PDE). The most well-known one is the relation between the streamfunction ψ and the vertical vorticity ζ (also known as Poisson equation):

$$\nabla^2\psi = \frac{\partial^2\psi}{\partial y^2} + \frac{\partial^2\psi}{\partial y^2} = \zeta$$

Once ζ is given as a known, one need to get the unknown ψ with proper boundary conditions, which is essentially an inversion problem (Figure 1). Many geophysical fluid dynamical (GFD) problems can be described by such a balanced model, which is generally of second (or fourth) order in spatial derivatives and do not depend explicitly on time. Therefore, it is also known as a steady-state model. Early scientists tried to find analytical solutions by simplified the parameters of these models (e.g., assuming constant coefficients). Nowadays, with the new developments in numerical algorithms and parallel-computing programming, one may need a modern solver, written in a popular programming language like `Python`, to invert all these models in a numerical fashion. More specifically, the following needs should be satisfied:

- **A unified numerical solver:** It can solve all the classical balanced GFD models, even in a domain with irregular boundaries like the ocean. New models can also be adapted straightforwardly to fit the solver;
- **Thinking and coding in equations:** Users focus naturally on the key inputs and outputs of the GFD models, just like thinking of the knowns and unknowns of the PDEs;
- **Flexible parameter specification:** Coefficients of the models can be either constant, 1D vector, or ND array. This allows an easy reproduce of early simplified results and also an extension to more general/realistic results;
- **Fast and efficient:** The algorithm should be fast and efficient. Also, the codes can be compiled first and then executed as fast as `C` or `FORTRAN`, instead of executed in the slow pure `Python` interpreter. In addition, it can leverage the multi-core and out-of-core computational capabilities of modern computers;

`xinvert` is then designed to satisfy all the above needs, based on the ecosystem of `Python`.

The Journal of Open Source Software

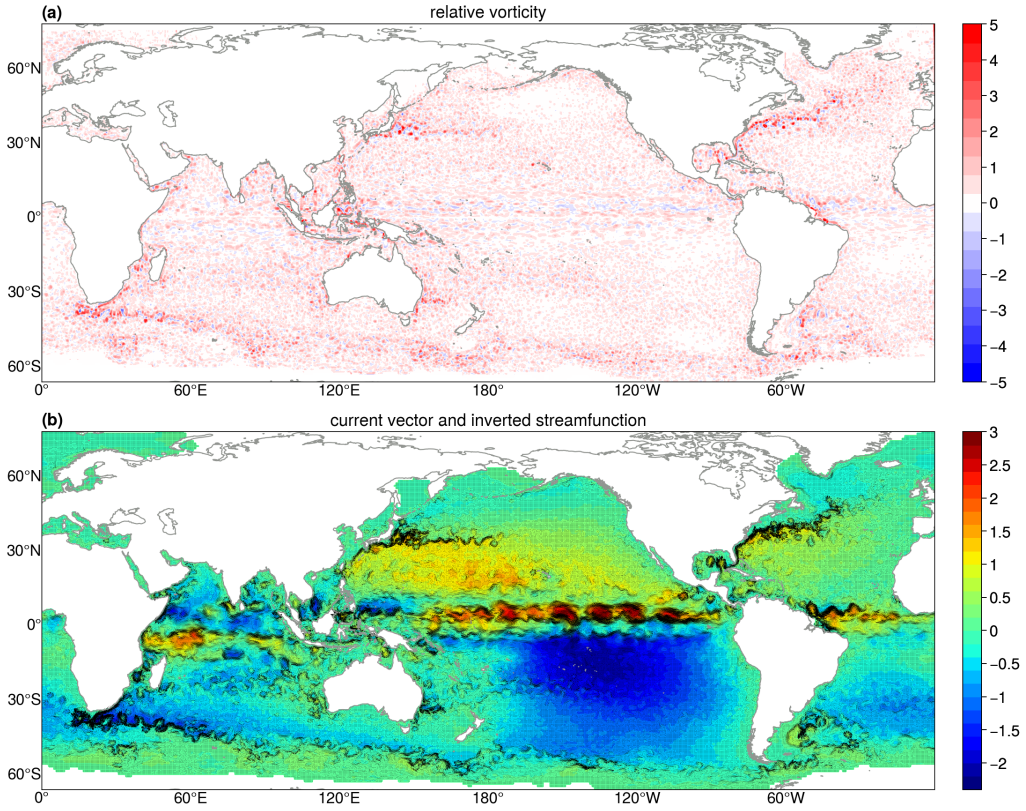

Figure 1: (a) Vertical relative vorticity ζ and (b) the inverted streamfunction ψ (shading) with current vector superimposed. Note the irregular boundaries over the global ocean.

Mathematics

This package, `xinvert`, is designed to invert or solve the following PDE in an abstract form:

$$L\left(\psi\right) = F \tag{1}$$

where L is a second-order partial differential operator, ψ the unknown to be inverted for, and F a prescribed forcing function. There could be also some coefficients or parameters in the definition of L, which should be specified before inverting ψ.

For the 2D case, the **general form** of Eq. (1) is:

$$L\left(\psi\right) \equiv A_1\frac{\partial^2\psi}{\partial y^2} + A_2\frac{\partial^2\psi}{\partial y\partial x} + A_3\frac{\partial^2\psi}{\partial x^2} + A_4\frac{\partial\psi}{\partial y} + A_5\frac{\partial\psi}{\partial x} + A_6\psi = F \tag{2}$$

where coefficients $A_1 - A_6$ are all known variables. When the condition $4A_1A_3 - A_2^2 > 0$ is met everywhere in the domain, the above equation is an elliptic-type equation. In this case, one can invert ψ using the **successive over relaxation (SOR)** iteration method. When $4A_1A_3 - A_2^2 = 0$ or $4A_1A_3 - A_2^2 < 0$, it is a parabolic or hyperbolic equation. In either case, SOR would *fail* to converge to the solution.

Sometimes the **general form** of Eq. (2) can be transformed into the **standard form** (i.e., standarization):

$$L\left(\psi\right) \equiv \frac{\partial}{\partial y}\left(A_1\frac{\partial \psi}{\partial y} + A_2\frac{\partial \psi}{\partial x}\right) + \frac{\partial}{\partial x}\left(A_3\frac{\partial \psi}{\partial y} + A_4\frac{\partial \psi}{\partial x}\right) + A_5\psi = F \tag{3}$$

In this case, $A_1A_4 - A_2A_3 > 0$ should be met to insure its ellipticity. The elliptic condition has its own physical meaning in the problems of interest. That is, the system is in a steady (or balanced) state that is stable to any small perturbation.

Many problems in meteorology and oceanography can be cast into the forms of either Eq. (2) or Eq. (3). However, some of them are formulated in 3D case (like the QG-omega equation):

$$L\left(\psi\right) \equiv \frac{\partial}{\partial z}\left(A_1\frac{\partial \psi}{\partial z}\right) + \frac{\partial}{\partial y}\left(A_2\frac{\partial \psi}{\partial y}\right) + \frac{\partial}{\partial x}\left(A_3\frac{\partial \psi}{\partial x}\right) = F \tag{4}$$

or in **fourth-order** case (the Munk model):

$$\begin{aligned} L\left(\psi\right) \equiv\ & A_1\frac{\partial^4 \psi}{\partial y^4} + A_2\frac{\partial^4 \psi}{\partial y^2 \partial x^2} + A_3\frac{\partial^4 \psi}{\partial x^4} \\ & + A_4\frac{\partial^2 \psi}{\partial y^2} + A_5\frac{\partial^2 \psi}{\partial y \partial x} + A_6\frac{\partial^2 \psi}{\partial x^2} + A_7\frac{\partial \psi}{\partial y} + A_8\frac{\partial \psi}{\partial x} + A_9\psi = F \end{aligned} \tag{5}$$

So we implements four basic solvers to take into account the above four Eqs. (2)-(5) cases. Most of the problems fit one of these four types of solver. However, it is **NOT** clear which form, the genral form Eq. (2) or the standard form Eq. (3), is preferred for the inversion if a problem can be cast into either one.

Summary

`xinvert` is an open-source and uesr-friendly `Python` package that enables GFD scientists or interested amateurs to solve all possible GFD problems in a numerical fashion. With the ecosystem of open-source `Python` packages, in particular `xarray` (Hoyer & Hamman, 2017), `dask` (Rocklin, 2015), and `numba` (Lam et al., 2015), it is able to satisfy the above requirements:

- All the classical balanced GFD models can be inverted by this unified numerical solver;
- User APIs (Table 1) are very close to the equations: unknowns are on the left-hand side of the equal sign =, whereas the known forcing functions are on its right-hand side (other known coefficients are also on the left-hand side but are passed in through `mParams`);
- Passing a single `xarray.DataArray` is usually enough for the inversion. Coordinates information is already encapsulated and thus reducing the length of the parameter list. In addition, paramters in `mParams` can be either a constant, or varying with a specific dimension (like Coriolis parameter f), or fully varying with space and time, due to the use of `xarray`'s (Hoyer & Hamman, 2017) broadcasting capability;
- This package leverages `numba` (Lam et al., 2015) and `dask` (Rocklin, 2015) to support Just-In-Time (JIT) compilation, multi-core, and out-of-core computations, and therefore greatly increases the speed and efficiency of the inversion.

Here we summarize the inversion problems in meteorology and oceanography into Table 1. The table can be extended further if one finds more problems that fit the abstract form of Eq. (1).

Table 1: Classical inversion problems in GFD. The model names, equations, typical references and function calls are listed

Problem names and equations	Function calls
Horizontal streamfunction $\nabla^2\psi = \frac{\partial^2\psi}{\partial y^2} + \frac{\partial^2\psi}{\partial x^2} = \zeta_k$	`psi = invert_Poisson(vork, dims=['Y','X'], mParams=None)`
MOC streamfunction $\nabla^2\psi = \frac{\partial^2\psi}{\partial z^2} + \frac{\partial^2\psi}{\partial y^2} = \zeta_i$	`psi = invert_Poisson(vori, dims=['Z','Y'], mParams=None)`
Walker streamfunction $\nabla^2\psi = \frac{\partial^2\psi}{\partial z^2} + \frac{\partial^2\psi}{\partial x^2} = \zeta_j$	`psi = invert_Poisson(vorj, dims=['Z','X'], mParams=None)`
Balanced mass field (Yuan et al., 2008) $\nabla^2\Phi = \frac{\partial^2\Phi}{\partial y^2} + \frac{\partial^2\Phi}{\partial x^2} = F$	`Phi = invert_Poisson(F, dims=['Y','X'], mParams=None)`
Geostrophic streamfunction $\frac{\partial}{\partial y}\left(f\frac{\partial\psi}{\partial y}\right) + \frac{\partial}{\partial x}\left(f\frac{\partial\psi}{\partial x}\right) = \nabla^2\Phi$	`psi = invert_geostrophic(LapPhi, dims=['Y','X'], mParams={f})`
Eliassen model (Eliassen, 1952) $\frac{\partial}{\partial p}\left(A\frac{\partial\psi}{\partial p} + B\frac{\partial\psi}{\partial y}\right) + \frac{\partial}{\partial y}\left(B\frac{\partial\psi}{\partial p} + C\frac{\partial\psi}{\partial y}\right) = F$	`psi = invert_Eliassen(F, dims=['Z','Y'], mParams={Angm, Thm})`
PV inversion for vortex (Hoskins et al., 1985) $\frac{\partial}{\partial\theta}\left(\frac{2\Lambda_0}{r^2}\frac{\partial\Lambda}{\partial\theta}\right) + \frac{\partial}{\partial r}\left(\frac{\Gamma g}{Qr}\frac{\partial\Lambda}{\partial r}\right) = 0$	`angM = invert_RefState(PV, dims=['Z','Y'], mParams={ang0, Gamma})`
PV inversion for QG flow $\frac{\partial}{\partial p}\left(\frac{f^2}{N^2}\frac{\partial\psi}{\partial p}\right) + \frac{\partial^2\psi}{\partial y^2} = q$	`psi = invert_PV2D(PV, dims=['Z','Y'], mParams={f, N2})`
Gill-Matsuno model (Gill, 1980; Matsuno, 1966) $A\frac{\partial^2\phi}{\partial y^2} + B\frac{\partial^2\phi}{\partial x^2} + C\frac{\partial\phi}{\partial y} + D\frac{\partial\phi}{\partial x} + E\phi = Q$	`phi = invert_GillMatsuno(Q, dims=['Y','X'], mParams={f, epsilon, Phi})`
Stommel-Munk model (Munk, 1950; Stommel, 1948) $A\nabla^4\psi - \frac{R}{D}\nabla^2\psi - \beta\frac{\partial\psi}{\partial x} = -\frac{\hat{\nabla}\cdot\vec{\tau}}{\rho_0 D}$	`psi = invert_StommelMunk(curl, dims=['Y','X'], mParams={A, R, D, beta, rho})`
QG-Omega equation (Hoskins et al., 1978) $\frac{\partial}{\partial p}\left(f^2\frac{\partial\omega}{\partial p}\right) + \nabla\cdot(S\nabla\omega) = F$	`w = invert_Omega(F, dims=['Z','Y','X'], mParams={f, S})`
3D ocean flow $\frac{\partial}{\partial p}\left(c_3\frac{\partial\psi}{\partial p}\right) + \nabla\cdot\left(c_1\nabla\psi - c_2\hat{\nabla}\psi\right) = F$	`psi = invert_3DFlow(F, dims=['Z','Y','X'], mParams={f, N2, epsilon})`
... more problems ...	

Usage

`xinvert` is designed in a functional-programming (FP) style. Users only need to import the function they are interested in and call it to get the inverted results (Table 1). Note that the calculation of the forcing function F (e.g., calculating the vorticity using velocity vector) on the right-hand side of the equation is **NOT** the core part of this package. But there is a `FiniteDiff` utility module with which finite difference calculus can be readily performed.

Acknowledgements

This work is jointly supported by the National Natural Science Foundation of China (42227901, 41931182, 41976023, 42106008), and the support of the Independent Research Project Program of State Key Laboratory of Tropical Oceanography (LTOZZ2102). The author gratefully acknowledge the use of the HPCC at the South China Sea Institute of ceanology, Chinese Academy of Sciences.

References

Eliassen, A. (1952). Slow thermally or frictionally controlled meridional circulation in a circular vortex. *Astrophisica Norvegica*, *5*(2), 19–60.

Gill, A. E. (1980). Some simple solutions for heat-induced tropical circulation. *Quarterly Journal of the Royal Meteorological Society*, *106*(449), 447–462. https://doi.org/10.1002/qj.49710644905

Hoskins, B. J., Draghici, I., & Davies, H. C. (1978). A new look at the ω-equation. *Quarterly Journal of the Royal Meteorological Society*, *104*(439), 31–38. https://doi.org/10.1002/qj.49710443903

Hoskins, B. J., McIntyre, M. E., & Robertson, A. W. (1985). On the use and significance of isentropic potential vorticity maps. *Quarterly Journal of the Royal Meteorological Society*, *111*(470), 877–946. https://doi.org/10.1002/qj.49711147002

Hoyer, S., & Hamman, J. (2017). Xarray: ND labeled arrays and datasets in python. *Journal of Open Research Software*, *5*(1). https://doi.org/10.5334/jors.148

Lam, S. K., Pitrou, A., & Seibert, S. (2015). Numba: A LLVM-based python JIT compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 1–6. https://doi.org/10.1145/2833157.2833162

Matsuno, T. (1966). Quasi-geostrophic motions in the equatorial area. *Journal of the Meteorological Society of Japan*, *44*(1), 25–43. https://doi.org/10.2151/jmsj1965.44.1_25

Munk, W. H. (1950). On the wind-driven ocean circulation. *Journal of the Atmospheric Sciences*, *7*(2), 80–93. https://doi.org/10.1175/1520-0469(1950)007%3C0080:OTWDOC%3E2.0.CO;2

Rocklin, M. (2015). Dask: Parallel Computation with Blocked algorithms and Task Scheduling. *Proceedings of the 14th Python in Science Conference*, 126–132. https://doi.org/10.25080/majora-7b98e3ed-013

Stommel, H. (1948). The westward intensification of wind-driven ocean currents. *Eos, Transactions American Geophysical Union*, *29*(2), 202–206. https://doi.org/10.1029/TR029i002p00202

Yuan, Z., Wu, J., Cheng, X., & Jian, M. (2008). The derivation of a numerical diagnostic model for the forcing of the geopotential. *Quarterly Journal of the Royal Meteorological Society*, *134*(637), 2067–2078. https://doi.org/10.1002/qj.337