qgs: A flexible Python framework of reduced-order multiscale climate models

Jonathan Demaeyer[1], **Lesley De Cruz**[1], **and Stéphane Vannitsem**[1]

1 Royal Meteorological Institute of Belgium, Avenue Circulaire, 3, 1180 Brussels, Belgium

DOI: 10.21105/joss.02597

Software

- Review
- Repository
- Archive

Editor: Alice Harpole

Reviewers:

- @eviatarbach
- @sadielbartholomew

Submitted: 05 August 2020
Published: 24 December 2020

Summary

qgs is a a Python implementation of a set of idealized reduced-order models representing atmospheric mid-latitude variability. It consists of a two-layer *q*uasi-*g*eostrophic *s*pectral (qgs) model of the atmosphere on a beta-plane, coupled either to a simple land surface or to a shallow-water ocean. The model's dynamical fields include the atmospheric and oceanic streamfunction and temperature fields, and the land temperature field.

- In the case where it is coupled to an ocean, it reproduces the Modular Arbitrary-Order Ocean-Atmosphere Model (MAOOAM), described in De Cruz et al. (2016). In Vannitsem et al. (2015), a 36-variable configuration of this model was shown to reproduce a low-frequency variability (LFV) typical of the coupled ocean-atmosphere system. This coupling consists in both mechanical and heat exchange interactions between the two components. The model has already been used in different contexts, in particular for data assimilation (Penny et al., 2019; Tondeur et al., 2020), and predictability studies (Vannitsem et al., 2019; Vannitsem & Duan, 2020)
- In the case of a land surface coupling, it emulates the model proposed in Reinhold & Pierrehumbert (1982) and Cehelsky & Tung (1987) with a simple thermal relaxation toward a climatological temperature and a mechanical coupling due to the friction between the land and the atmosphere. It can also emulate the model proposed in Li et al. (2018), with mechanical coupling and heat exchange. In addition, the number of dynamical spectral modes can be configured by the user, as is the case for the MAOOAM model.

In the qgs framework, the partial differential equations (PDEs) that govern the time evolution of its fields are projected on a basis of functions defined on its spatial domain. This kind of decomposition transforms the PDEs into a set of ordinary differential equations (ODEs) which can then be solved with the usual integration techniques. Presently in qgs, the functions of the basis are chosen amongst the orthogonal Fourier modes compatible with the boundary conditions of each subcomponent of the system, namely the atmosphere, and the ocean or the land surface coupled to it. A future development is planned that will enable the user to specify the basis of functions for each component, depending on the required boundary conditions.

The model implementation consists of submodules to set up the model's parameters and to compute the tensor that defines the coefficients of the system of ODEs[1]. This tensor is used by the code to compute the tendencies function and its Jacobian matrix. These functions can then be fed to the qgs built-in Runge-Kutta integrator or to another integrator implemented by the user. As an example, the usage of the Julia `DifferentialEquations.jl` (Rackauckas & Nie, 2017) integration package through the Python `diffeqpy` (Rackauckas & Arakaki, 2020)

[1]More details about the implementation can be found in De Cruz et al. (2016) and in the *Code Description* section of the included documentation.

package is provided. The tangent linear and adjoint models (Kalnay, 2003) are also available and allow one to easily conduct data assimilation and linear sensitivity analysis experiments.

The model implementation uses NumPy (Harris et al., 2020; Oliphant, 2006) and SciPy (Virtanen et al., 2020) for arrays and computations support, as well as Numba (Lam et al., 2015) and sparse (Sparse developers, 2020) to considerably accelerate the tensor products computation used to compute the tendencies.

Statement of need

In atmospheric and climate sciences, research and development is often first conducted with a simple idealized system like the Lorenz-N models ($N \in \{63, 84, 96\}$) (Edward N. Lorenz, 1996, 1984; Edward N. Lorenz, 1963) which are toy models of atmospheric variability. The first two models are heavily truncated systems (3-variable) describing the very large synoptic-scale dynamics of the single-component atmosphere, that neglect the interaction with other components of the climate system and with smaller scales. The third one is based on reasonable heuristic assumptions on the spatial dynamics along a latitude, which may however lead to unrealistic statistical features.

Reduced-order spectral quasi-geostrophic models of the atmosphere with a large number of modes offer better representations of the dry atmospheric dynamics (O'Brien & Branscome, 1989). The dynamics thus obtained allow one to identify typical features of the atmospheric circulation, such as blocked and zonal circulation regimes, and low-frequency variability. However, these models are less often considered in literature, despite their demonstration of more realistic behavior.

qgs aims to popularize these systems by providing a fast and easy-to-use Python framework for researchers and teachers to integrate this kind of model. For an efficient handling of the model by users, its documentation is conceived such that its equations and parameters are explained and linked to the code. In the future, its development will be done in a modular fashion which enables the connection of the atmosphere to various other subsystems and the use of built-in and external toolboxes.

The choice to use Python was specifically made to facilitate its use in Jupyter Notebooks (Kluyver et al., 2016) and with the multiple recent machine learning libraries that are available in this language.

State of the field

Other software might interest the reader in need of an easy-to-use idealized atmospheric model.

- MAOOAM: The Modular Arbitrary-Order Ocean-Atmosphere Model, a coupled ocean-atmosphere model included in qgs (De Cruz & Demaeyer, 2020). Code available in Lua, Fortran and Python.
- q-gcm: A mid-latitude grid-based quasi-geostrophic ocean-atmosphere model with two oceanic layers. Code in Fortran, interface in Python (Hogg et al., 2014).
- pyqg: A pseudo-spectral Python solver for quasi-geostrophic systems (Jansen et al., 2019). Allow one to create and solve multiple-layers quasi-geostrophic systems.
- Isca: A research General Circulation Model (GCM) to simulate global dynamics. Written in Fortran and largely configurable with Python scripts, with internal coding changes required for non-standard cases (Isca development team, University of Exeter, 2020; Vallis et al., 2018).

qgs distinguishes itself from these other models by the combination of a simplified and configurable geometry, a spectral discretization, an efficient numerical implementation of the ODE system by a sparse tensor multiplication, and the availability of the tangent linear and adjoint models. As such it is very suitable to quickly simulate very long time periods while capturing key aspects of the climate dynamics at mid-latitudes.

The mechanically coupled atmosphere-land configuration of qgs was used to test new ideas using response theory to adapt statistical postprocessing schemes to model changes (Demaeyer & Vannitsem, 2020). The MAOOAM model configuration of qgs was recently considered to perform strongly-coupled data assimilation experiments in the ocean-atmosphere system (Carrassi et al., 2020).

Performance

The performance of the qgs MAOOAM implementation has been benchmarked against the Lua and Fortran implementations of this model (De Cruz & Demaeyer, 2020). This comparison was done on a recent Intel CPU with 12 cores, with two different model resolutions: one used in Vannitsem et al. (2015) and one truncated at the wavenumber 6 for both the oceanic and atmospheric components. The former leads to a 36-dimensional system of ODEs while the latter is higher-dimensional, using 228 variables.

In both cases, all the different code implementations have been initialized with the same initial data and parameters, except for the length of the trajectory being computed. The low-dimensional system was integrated for 10^7 timeunits (roughly ~ 1850 years) while the higher-dimensional one was integrated over 10^6 timeunits (~ 185 years). In the case of the Fortran implementation, two different compilers (GNU Gfortran and Intel Ifort) with two different levels of optimization (O2 and O3) have been tested, but no significant differences between these compilers and options were found. In addition, two different built-in integration modules of qgs have been considered: a non-parallel integrator located in the module `integrators.integrate` and a parallel one located in the module `integrators.integrator`. The latter can integrate multiple trajectories simultaneously, but for the purpose of the benchmark, only one trajectory was computed with it, the other implementations being non-parallel.

The results of this benchmark are depicted on Figure 1 and show that qgs, while not the fastest implementation of MAOOAM available, is a fair competitor. The time difference is in general not greater than a factor 5 and tends to be less for high-dimensional model configurations, with an integration time roughly the same as the Lua implementation. We note that there is also a significant difference between the parallel and non-parallel implementation of qgs, but this difference also seems to vanish for higher-resolution model configurations. In any case, the parallel integrator of qgs can straightforwardly integrate multiple trajectories simultaneously and therefore has an advantage over the non-parallel one (provided that multiple CPU cores are available). A final remark is that the initial Python version of MAOOAM (found in De Cruz & Demaeyer, 2020) takes 283 minutes to integrate the low-resolution model configuration (not shown).

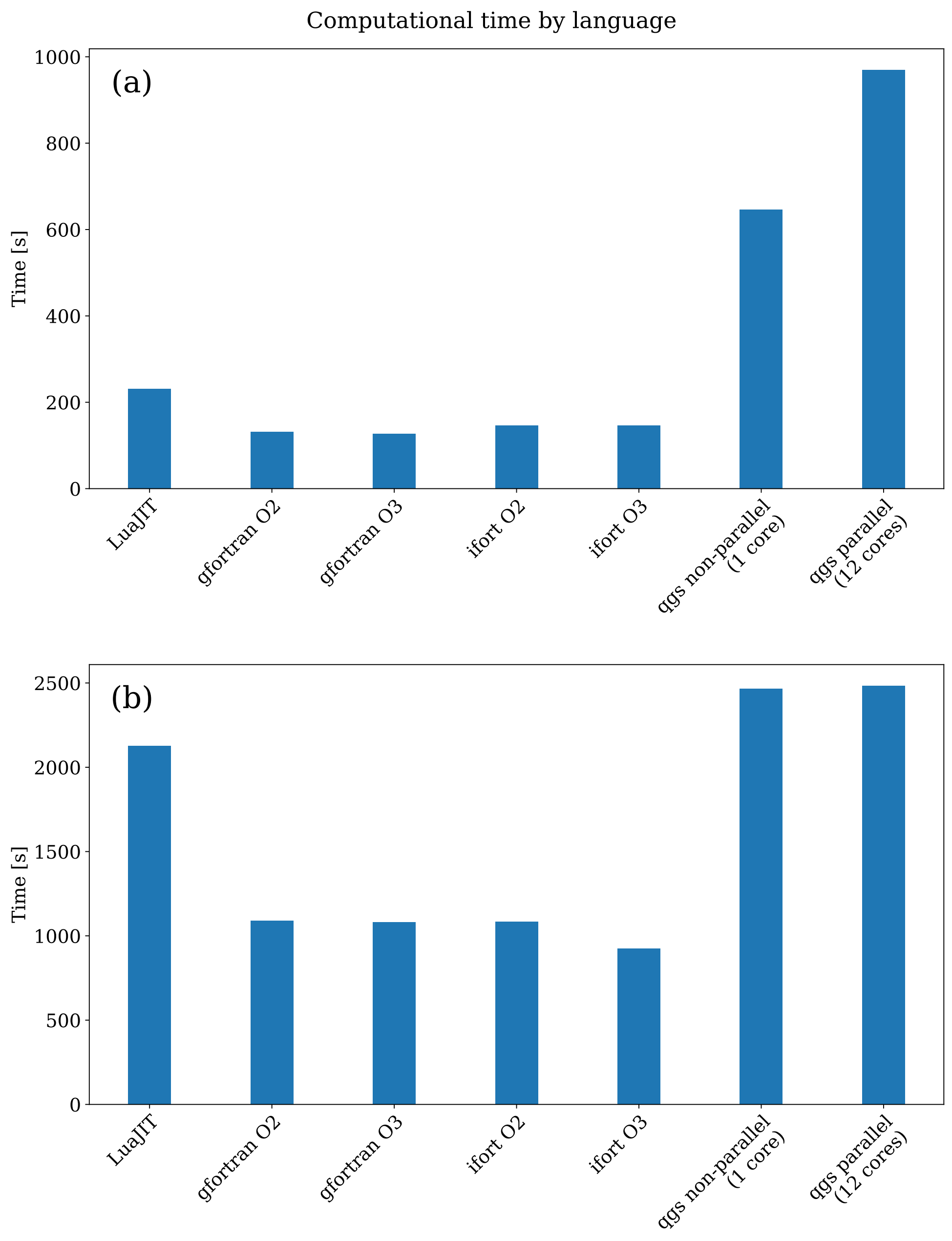

Figure 1: Computational times in seconds of different MAOOAM implementations: (a) time to compute a 10^7 timeunits trajectory with a low-order model configuration (36 variables). (b) time to compute a 10^6 timeunits trajectory with a higher-order model configuration (228 variables).

In conclusion, qgs is a sufficiently fast Python implementation as compared to the other implementations of the MAOOAM model. In addition, it has the benefit of being more flexible, extensible, and easier to use in the general Python scientific ecosystem.

Acknowledgements

This research has been partly supported by EUMETNET (Postprocessing module of the NWP Cooperation Programme).

References

Carrassi, A., Bocquet, M., Demaeyer, J., Grudzien, C., Raanes, P., & Vannitsem, S. (2020). Data assimilation for chaotic dynamics. *arXiv Preprint arXiv:2010.07063*.

Cehelsky, P., & Tung, K. K. (1987). Theories of multiple equilibria and weather regimes - A critical reexamination. Part II: Baroclinic two-layer models. *Journal of the Atmospheric Sciences*, *44*(21), 3282–3303. https://doi.org/10.1175/1520-0469(1987)044

De Cruz, L., & Demaeyer, J. (2020). MAOOAM: Modular arbitrary-order ocean-atmosphere model. In *GitHub repository*. GitHub. https://github.com/Climdyn/MAOOAM

De Cruz, L., Demaeyer, J., & Vannitsem, S. (2016). The modular arbitrary-order ocean-atmosphere model: MAOOAM v1.0. *Geoscientific Model Development*, *9*(8), 2793–2808. https://doi.org/10.5194/gmd-9-2793-2016

Demaeyer, J., & Vannitsem, S. (2020). Correcting for model changes in statistical post-processing – an approach based on response theory. *Nonlinear Processes in Geophysics*, *27*(2), 307–327. https://doi.org/10.5194/npg-27-307-2020

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., R'ıo, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hogg, A., Dewar, B., Blundell, J., & Chapman, C. (2014). *Q-gcm*. http://q-gcm.org/

Isca development team, University of Exeter. (2020). Isca. In *GitHub repository*. GitHub. https://github.com/ExeClim/Isca

Jansen, M., Abernathey, R., Rocha, C., & Poulin, F. (2019). Pyqg. In *GitHub repository*. GitHub. https://github.com/pyqg/pyqg

Kalnay, E. (2003). *Atmospheric modeling, data assimilation and predictability*. Cambridge university press.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., & Willing, C. (2016). *Jupyter notebooks - a publishing format for reproducible computational workflows* (F. Loizides & B. Schmidt, Eds.; pp. 87–90). IOS Press. https://doi.org/10.3233/978-1-61499-649-1-87

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

Li, D., He, Y., Huang, J., Bi, L., & Ding, L. (2018). Multiple equilibria in a land–atmosphere coupled system. *Journal of Meteorological Research*, *32*(6), 950–973. https://doi.org/10.1007/s13351-018-8012-y

Lorenz, Edward N. (1996). Predictability: A problem partly solved. *Proc. Seminar on Predictability*, *1*.

Lorenz, Edward N. (1963). Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, *20*(2), 130–141. https://doi.org/10.1175/1520-0469(1963)020%3C0130:DNF%3E2.0.CO;2

Lorenz, Edward N. (1984). Irregularity: A fundamental property of the atmosphere. *Tellus A*, *36*(2), 98–110. https://doi.org/10.1111/j.1600-0870.1984.tb00230.x

O'Brien, E., & Branscome, L. (1989). Minimal modeling of the extratropical general circulation. *Tellus A: Dynamic Meteorology and Oceanography*, *41*(4), 292–307. https://doi.org/10.3402/tellusa.v41i4.11842

Oliphant, T. E. (2006). *A guide to NumPy* (Vol. 1). Trelgol Publishing USA.

Penny, S., Bach, E., Bhargava, K., Chang, C.-C., Da, C., Sun, L., & Yoshida, T. (2019). Strongly coupled data assimilation in multiscale media: Experiments using a quasi-geostrophic coupled model. *Journal of Advances in Modeling Earth Systems*, *11*(6), 1803–1829. https://doi.org/10.1029/2019MS001652

Rackauckas, C., & Arakaki, T. (2020). Diffeqpy. In *GitHub repository*. GitHub. https://github.com/SciML/diffeqpy

Rackauckas, C., & Nie, Q. (2017). Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, *5*(1). https://doi.org/10.5334/jors.151

Reinhold, B., & Pierrehumbert, R. (1982). Dynamics of weather regimes: Quasi-stationary waves and blocking. *Monthly Weather Review*, *110*, 1105–1145. https://doi.org/10.1175/1520-0493(1982)110

Sparse developers. (2020). Sparse. In *GitHub repository*. GitHub. https://github.com/pydata/sparse

Tondeur, M., Carrassi, A., Vannitsem, S., & Bocquet, M. (2020). On temporal scale separation in coupled data assimilation with the ensemble kalman filter. *Journal of Statistical Physics*, *179*, 1161–1185. https://doi.org/10.1007/s10955-020-02525-z

Vallis, G. K., Colyer, G., Geen, R., Gerber, E., Jucker, M., Maher, P., Paterson, A., Pietschnig, M., Penn, J., & Thomson, S. I. (2018). Isca, v1.0: A framework for the global modelling of the atmospheres of earth and other planets at varying levels of complexity. *Geoscientific Model Development*, *11*(3), 843–859. https://doi.org/10.5194/gmd-11-843-2018

Vannitsem, S., Demaeyer, J., De Cruz, L., & Ghil, M. (2015). Low-frequency variability and heat transport in a low-order nonlinear coupled ocean–atmosphere model. *Physica D: Nonlinear Phenomena*, *309*, 71–85. https://doi.org/10.1016/j.physd.2015.07.006

Vannitsem, S., & Duan, W. (2020). On the use of near-neutral backward lyapunov vectors to get reliable ensemble forecasts in coupled ocean-atmosphere systems. *Climate Dynamics*. https://doi.org/10.1007/s00382-020-05313-3

Vannitsem, S., Solé-Pomies, R., & De Cruz, L. (2019). Routes to long-term atmospheric predictability in reduced-order coupled ocean–atmosphere systems: Impact of the ocean basin boundary conditions. *Quarterly Journal of the Royal Meteorological Society*, *145*(723), 2791–2805. https://doi.org/10.1002/qj.3594

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Jarrod Millman, K., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … Contributors, S. 1. 0. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2