SoftwareX 17 (2022) 100982

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Quail: A lightweight open-source discontinuous Galerkin code in Python for teaching and prototyping

Eric J. Ching, Brett Bornhoft *, Ali Lasemi, Matthias Ihme

Department of Mechanical Engineering, Stanford University, Stanford, CA 94305, USA

ARTICLE INFO

Article history:
Received 25 January 2021
Received in revised form 2 December 2021
Accepted 11 January 2022

Keywords:
Discontinuous Galerkin method
High-order methods
Python

ABSTRACT

In this paper, we present Quail, a lightweight discontinuous Galerkin solver written in Python. The aim of this code is to serve not only as a teaching tool for newcomers to the rapidly growing field, but also as a prototyping platform for testing algorithms, physical models, and other features in the discontinuous Galerkin framework. Code readability, modularity, and ease of use are emphasized. Currently, Quail solves first- and second-order partial differential equations on 1D and 2D unstructured meshes. A variety of time stepping schemes, quadrature rules, basis types, equation sets, and other features are included. The structure and capabilities of the code, as well as representative examples involving propagation of a 2D isentropic vortex and a 2D Riemann problem with a gravity source term, will be discussed.

Code metadata

Current code version	v1.0.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00020
Code Ocean compute capsule	None
Legal Code License	GNU General Public License v3.0
Code versioning system used	git
Software code languages, tools, and services used	Python 3.7
Compilation requirements, operating environments & dependencies	NumPy [1], Matplotlib [2], SciPy [3]
If available Link to developer documentation/manual	https://github.com/IhmeGroup/quail/blob/main/docs/documentation.pdf
Support email for questions	bornhoft@stanford.edu

1. Motivation and significance

Numerical solutions to partial differential equations are relevant to many areas of science and engineering, including fluid dynamics, solid mechanics, electrodynamics, and astrophysics. A variety of numerical methods, such as finite difference and finite volume schemes, can be used to discretize the governing equations describing these problems. In recent years, high-order discontinuous Galerkin (DG) methods have gained considerable interest [4,5]. These methods combine aspects of classical finite volume and finite element schemes. The global solution is approximated using piecewise discontinuous polynomials, resulting in multiple degrees of freedom per element. Discontinuities between elements are accounted for with numerical flux functions. Benefits of DG schemes include high-order accuracy (typically defined as greater than second-order), desirable dissipation and dispersion properties, geometric flexibility, and suitability for *hp*-adaptation, where *h* refers to the mesh and *p* refers to the order of accuracy. In addition, DG methods can achieve very good scalability and efficiency on high-performance computing systems. Encouraging performance has been demonstrated in aerodynamics [6,7], multiphase flows [8,9], plasma physics [10,11], astrophysics [12,13], and solid mechanics [14,15]. However, DG schemes and related high-order methods, such as flux reconstruction [16] and spectral difference schemes [17,18], are generally less robust and more memory-intensive than low-order schemes. Furthermore, curved meshes, often required for these high-order methods, are difficult to generate, and nontrivial extensions can be required to account for additional physics. These factors currently hinder widespread use of DG schemes for industrial applications.

* Corresponding author.
E-mail address: bornhoft@stanford.edu (Brett Bornhoft).

https://doi.org/10.1016/j.softx.2022.100982

Another difficulty associated with DG schemes is that they are considered more complicated to learn and implement than conventional low-order methods [19]. This barrier to entry can not only discourage engineers and scientists from entering the field, but also impede progress in both implementation and algorithmic development. Reliable learning resources can help lower this barrier. Good textbooks are available that describe the theory and application of DG schemes [5,20]. There are also a handful of open-source codes on DG and related high-order methods [21–26]. However, these are typically large, production-level codes, which can be overwhelming for newcomers and difficult to set up and run. The textbook by Hesthaven and Warburton [5] includes a well-known set of routines written in MATLAB and C++, but it can be difficult to add new features in a modular fashion.

In this paper, we present Quail, a lightweight DG code designed to address the above issues. The code is written in Python because it is open-source (unlike MATLAB), well-documented, suited for object-oriented programming, growing in popularity, and easier to read and use than lower-level languages such as C, C++, and Fortran. The primary objective of Quail is to serve as a teaching and prototyping tool. To support this goal, we focus on code modularity, clarity, and straightforward setup and usage. The simplicity of Quail allows users to easily peruse, understand, and add to the code without the intricacies of large codebases.

The remainder of this paper is organized as follows. Section 2 provides a brief overview of the discretization. Section 3 describes the software architecture and functionalities, followed by an illustrative example consisting of the propagation of a 2D isentropic vortex. The final two sections comprise a discussion of the impact of this code and concluding remarks, respectively.

2. Mathematical background

Quail solves linear and nonlinear systems of PDEs of the following form:

$$\partial_t \boldsymbol{U} + \nabla \cdot \boldsymbol{F} = \boldsymbol{S}, \tag{1}$$

where $\boldsymbol{U}$ is the vector of state variables, $\boldsymbol{F}$ is the flux, and $\boldsymbol{S}$ is the source term. For brevity, this section assumes first-order PDEs, although Quail can also solve second-order PDEs. Let Ω denote the computational domain, which is partitioned into N_e non-overlapping discrete elements such that $\Omega = \cup_{e=1}^{N_e} \Omega_e$. $\partial\Omega_e$ is the boundary of element Ω_e. The approximation to the global solution, $\boldsymbol{U}_h \simeq \boldsymbol{U}$, can be expanded as

$$\boldsymbol{U}_h = \oplus_{e=1}^{N_e} \boldsymbol{U}_h^e, \tag{2}$$

where $\boldsymbol{U}_h^e$ is the local discrete solution,

$$\boldsymbol{U}_h^e(\boldsymbol{x}, t) = \sum_{n=1}^{N_b} \widetilde{\boldsymbol{U}}_n^e(t) \phi_n(\boldsymbol{x}). \tag{3}$$

$\widetilde{\boldsymbol{U}}_n^e(t)$ is the nth vector of basis coefficients, and ϕ_n is the nth basis polynomial. Common choices for the basis include Lagrange polynomials and Legendre polynomials. The nominal order of accuracy in space for smooth solutions is $p+1$, where p is the order of the polynomial approximation.

To solve for the local discrete solution, we require $\boldsymbol{U}_h^e$ to satisfy

$$\int_{\Omega_e} \phi_m \partial_t \boldsymbol{U}_h^e d\Omega + \int_{\Omega_e} \phi_m \nabla \cdot \boldsymbol{F}(\boldsymbol{U}_h^e) d\Omega = \int_{\Omega_e} \phi_m \boldsymbol{S}(\boldsymbol{U}_h^e) d\Omega \qquad \forall\, \phi_m. \tag{4}$$

Inserting Eq. (3) into the first term in Eq. (4) allows us to write

$$\int_{\Omega_e} \phi_m \partial_t \boldsymbol{U}_h^e d\Omega = \sum_{n=1}^{N_b} d_t \widetilde{\boldsymbol{U}}_n^e(t) \int_{\Omega_e} \phi_m \phi_n d\Omega = \sum_{n=1}^{N_b} d_t \widetilde{\boldsymbol{U}}_n^e(t) M_{mn}^e, \tag{5}$$

where $M_{mn}^e = \int_{\Omega_e} \phi_m \phi_n d\Omega$ represents the element-local mass matrix.

Integration by parts is performed on the second term in Eq. (4), yielding

$$\int_{\Omega_e} \phi_m \nabla \cdot \boldsymbol{F}(\boldsymbol{U}_h^e) d\Omega = -\int_{\Omega_e} \nabla \phi_m \cdot \boldsymbol{F}(\boldsymbol{U}_h^e) d\Omega + \oint_{\partial\Omega_e} \phi_m \widehat{\boldsymbol{F}}(\boldsymbol{U}_h^{e+}, \boldsymbol{U}_h^{e-}, \widehat{\boldsymbol{n}}) d\Gamma, \tag{6}$$

where $\widehat{\boldsymbol{n}}$ is the outward-pointing unit normal vector, $(\cdot)^+$ and $(\cdot)^-$ denote interior and exterior information on $\partial\Omega_e$, respectively, and $\widehat{\boldsymbol{F}}$ is the numerical flux function.

Combining Eqs. (4), (5), and (6) gives the following semidiscrete form:

$$\sum_{n=1}^{N_b} d_t \widetilde{\boldsymbol{U}}_n^e(t) M_{mn}^e = \int_{\Omega_e} \nabla \phi_m \cdot \boldsymbol{F}(\boldsymbol{U}_h^e) d\Omega - \oint_{\partial\Omega_e} \phi_m \widehat{\boldsymbol{F}}(\boldsymbol{U}_h^{e+}, \boldsymbol{U}_h^{e-}, \widehat{\boldsymbol{n}}) d\Gamma + \int_{\Omega_e} \phi_m \boldsymbol{S}(\boldsymbol{U}_h^e) d\Omega. \tag{7}$$

The time derivative on the LHS can be treated using classical explicit time stepping schemes. The integrals are evaluated using numerical quadrature. Additional details on the discretization can be found in the documentation (Code metadata), as well as in Ref. [5].

3. Software description

Quail uses an object-oriented programming paradigm to achieve flexibility. Therefore, when a new feature is implemented, it is simple to inherit and build on existing class structures. Readability is attained by employing a Pythonic coding style and extensively using Python docstrings and comments. All mathematical operations are performed using the popular NumPy library [1], which enhances performance by utilizing fully vectorized array operations. For modularity, Quail is split into five Python packages, each responsible for encapsulating different aspects of the DG solution procedure. This software architecture is depicted in the diagram in Fig. 1, with specific capabilities listed in Table 1. A brief description of each package is given below:

- The `meshing` package handles the generation of the computational mesh and its data structures. High-order curved elements are supported. Built-in routines are provided to generate 1D and 2D regular meshes with quadrilateral and triangular elements. Periodicity can be imposed on user-specified boundaries as well. This package can also import meshes from the open-source meshing software Gmsh [27], which can generate curved unstructured meshes for complex geometries.
- The `numerics` package is responsible for the available numerical methods. These include various choices of quadrature rules for evaluating integrals, basis functions for the solution and geometric approximation, time-stepping schemes, and limiters and artificial viscosity for stabilization. The quadrature rules and basis types support segments in 1D and triangles and quadrilaterals in 2D, and there are both nodal and hierarchic bases available for all of these shapes. A Gauss–Lobatto collocated scheme, in which the solution nodes and the quadrature points are the same, is also included.

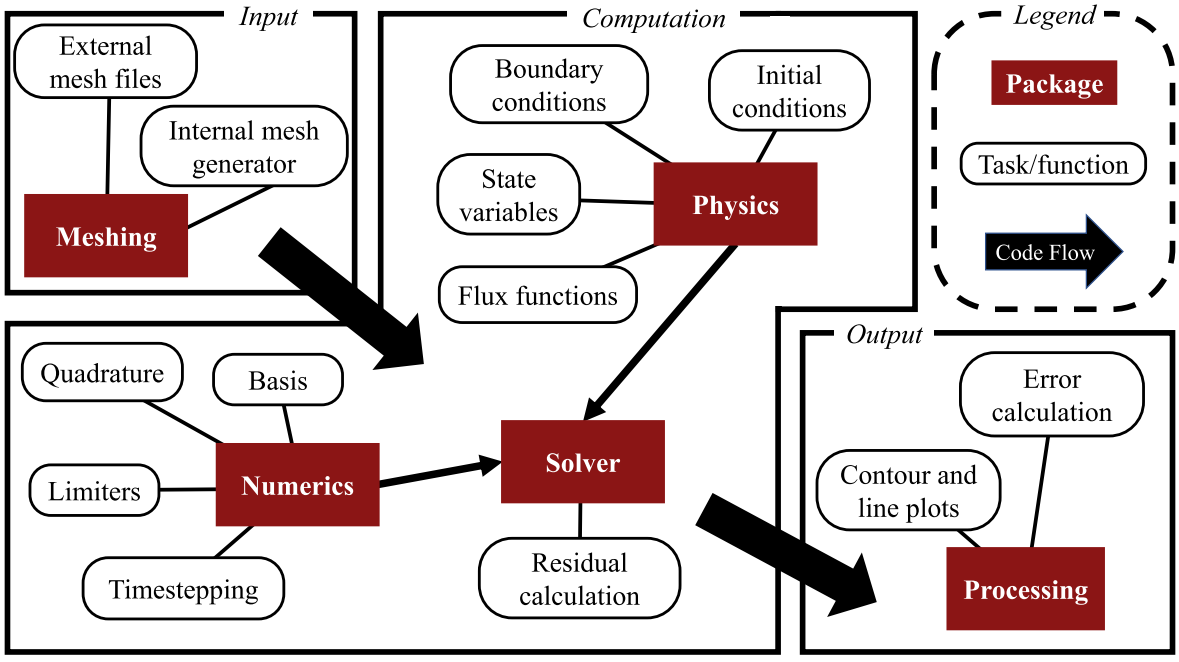


Fig. 1. Software architecture diagram depicting the tasks performed by each package, as well as the overall flow of the code.

- The `physics` package includes the equation sets and corresponding physical models. It contains the definitions of fluxes, initial conditions, boundary conditions, and source terms. Supported equation sets include constant scalar advection, the inviscid Burgers equation, and the compressible Euler equations. We have also recently added the capability to handle second-order PDEs such as the scalar advection–diffusion and compressible Navier–Stokes equations, which is undergoing further development.
- The `solver` package combines physics-related information stored in `physics` with the numerical algorithms in `numerics` to update the solution at each time step. The solution can be computed with not only the standard DG discretization, but also the ADERDG scheme [28,29], which is a space–time predictor–corrector method that allows for high-order accuracy in both space and time while maintaining robustness with stiff source terms.
- The `processing` package performs tasks on the solution data, such as computing error and generating visualizations, including 1D line plots, 2D contour plots, line probes, and animations. Matplotlib [2] subroutines are employed to allow for easy setup, implementation, and modification.

Solution data files for processing and simulation restarts are saved using the `pickle` module, which is part of the Python standard library. Simple subroutines allow for easy reading and writing of data. Entire Python objects can be saved with one line of code, a major advantage of this module. Another user-friendly feature of Quail is the option to define custom functions to perform case-specific data processing at each time step. These custom functions are written outside of the source code and therefore do not require a full understanding of the codebase, allowing for simple implementation. Also included is built-in support for continuous integration and a testing infrastructure that encompasses both functional and unit tests.

A suite of test cases can be found in the repository. These include a 1D damping sine wave, 2D advection of a Gaussian pulse, the Sod shock tube problem, a moving shockwave, steady flow over a bump, the steady inviscid Taylor–Green vortex [30], and others. The illustrative examples in Section 4 are also available.

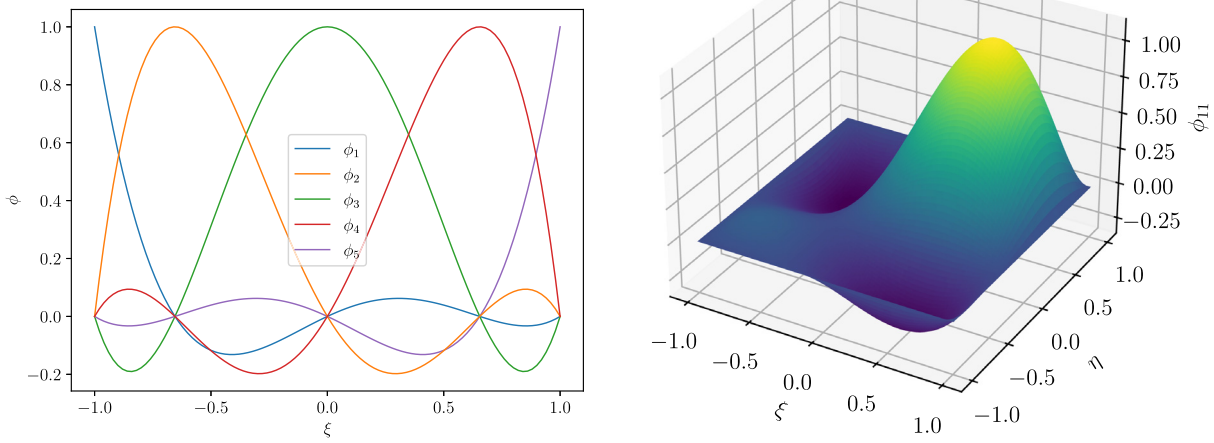
Additional learning tools are included in Quail. For example, the basis functions for segments, quadrilaterals, and triangles can be visualized, as shown in Fig. 2. Furthermore, dissipation and dispersion analysis for various polynomial orders can be performed. To illustrate, the dissipation and dispersion relations for $p = 1$ to $p = 7$ with an upwind flux are given in Fig. 3. More information on this analysis can be found in Ref. [5].

4. Illustrative examples

4.1. Isentropic vortex propagation

To illustrate the functionality of Quail, we present the numerical solution of a propagating isentropic vortex governed by the compressible Euler equations. This is a classical case for testing the order of accuracy of higher-order schemes. The input deck relies on Python dictionaries to prescribe solver and physics parameters, allowing for easy simulation setup. These input dictionaries are organized as follows: `TimeStepping`, `Numerics`, `Mesh`, `Physics`, `InitialCondition`, `ExactSolution`, `BoundaryConditions`, `SourceTerms`, `Output`, and `Restart`. Default parameters located in `src/defaultparams.py` allow users to modify only the necessary parameters for each individual case. Since Python scripts are used as input files, any Python functionality can be employed directly in the input deck. The input file for this case is shown below.

```
# Input deck for the setup of a
# propagating isentropic vortex
TimeStepping = {
  "FinalTime" : 1.0,
  "CFL" : 0.1,
  "TimeStepper" : "LSRK4",
}
Numerics = {
  "SolutionOrder" : 3,
  "SolutionBasis" : "LagrangeTri",
}
Mesh = {
  "ElementShape" : "Triangle",
  "NumElemsX" : 16,
  "NumElemsY" : 16,
  "xmin" : -5.,
```



(a) $p = 4$ Lagrange basis functions with Gauss-Lobatto nodes

(b) $p = 3$ Lagrange basis function with equidistant nodes

Fig. 2. Sample basis functions for (a) segments and (b) quadrilaterals.

```
  "xmax" : 5.,
  "ymin" : -5.,
  "ymax" : 5.,
}
Physics = {
  "Type" : "Euler",
  "ConvFluxNumerical" : "Roe",
  "GasConstant" : 1.,
}
InitialCondition = {
  "Function" : "IsentropicVortex",
}
ExactSolution = InitialCondition.copy()
d = {
  "BCType" : "StateAll",
  "Function" : "IsentropicVortex",
}
BoundaryConditions = {
  "x1" : d,
  "x2" : d,
  "y1" : d,
  "y2" : d,
}
```

The initial conditions are based on the work of Yu et al. [31], where velocity and temperature perturbations are superposed onto a uniform flow. The domain is a square of size $[-5, 5] \times [-5, 5]$. The exact solution corresponds to advection of the isentropic vortex at constant velocity. Fig. 4 shows density contours at a final time of one second on two different meshes: a $q = 1$ triangular mesh (Fig. 4(a)), where q refers to the order of the geometry approximation, and a curved $q = 2$ quadrilateral mesh (Fig. 4(c)). After the simulation is completed, the solver (optionally) searches for a post-processing script in the working directory. With the tools detailed in Section 3, users can create contour and line plots such as those in Fig. 4. The post-processing script for this case is as follows:

```
import processing.post as post
import processing.plot as plot
import processing.readwritedatafiles as
           readwritedatafiles
# Read data file
fname = "Data_final.pkl"
solver = readwritedatafiles.read_data_file(fname)
# Unpack
mesh = solver.mesh
physics = solver.physics
# Compute L2 error
post.get_error(mesh, physics, solver, "Density")
# Plot density contour
plot.prepare_plot(linewidth=0.1)
plot.plot_solution(mesh, physics, solver, "Density",
    legend_label="DG", include_mesh=True, regular_2D
           =True)
plot.save_figure(file_name='vortex', file_type='pdf'
           )
plot.show_plot()
```

This script reads in a `pickle` data file, unpacks the relevant objects from the solver, computes the L_2-error of density, and plots the density contour. This data can then be used to construct convergence plots such as that in Fig. 4(d), which shows that the expected convergence rate is obtained. Additional information on how to construct input files and post-processing scripts, including representative examples, can be found in the Quail repository (Code metadata).

4.2. 2D Riemann problem with gravity source term

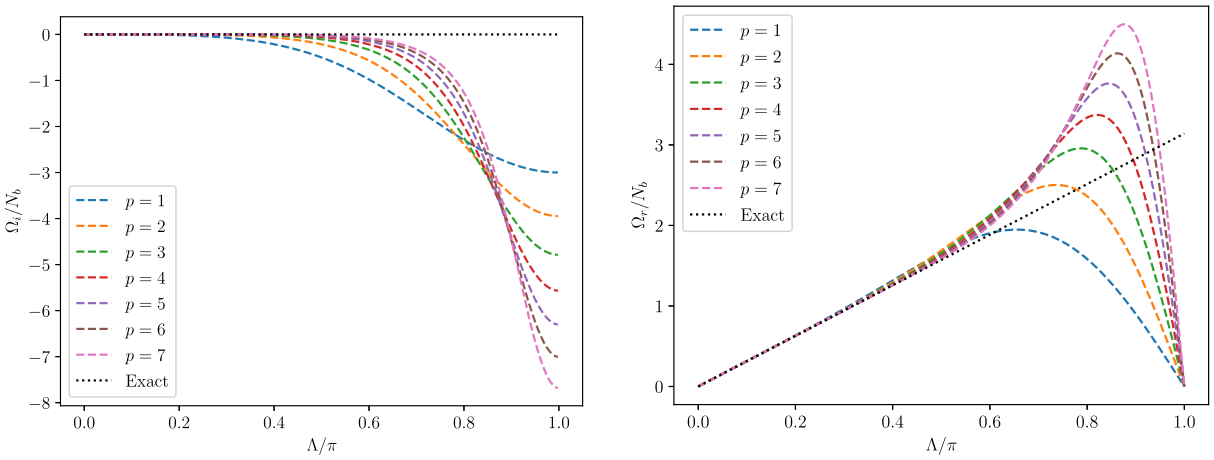
Quail's object-oriented framework makes adding source terms and numerical algorithms straightforward. Here, we illustrate the use of these functionalities by simulating a 2D Riemann problem with a gravity source term governed by the compressible Euler equations. As done in Ref. [32], the case is set up on a 2×2 domain with the following initial conditions:

$$(\rho, u, v, P) = \begin{cases} (7, -1, 0, 0.2)\,, & x \leq 1, \\ (7, 1, 0, 0.2)\,, & x > 1, \end{cases} \tag{8}$$

where ρ is the density, u and v are the x- and y-components of the velocity, respectively, and P is the pressure. The source term is given as $\boldsymbol{S} = [0, 0, -\rho g, -\rho v g]^T$, with $g = 1$. Under these conditions, both pressure and density can approach non-physical negative values. To improve robustness, we employ the positivity-preserving limiter by Zhang and Shu [33]. The polynomial order is $p = 1$. In Fig. 5, we show numerical results of the Riemann problem at $t = 0.6$ s, which are comparable to those in the literature [32].

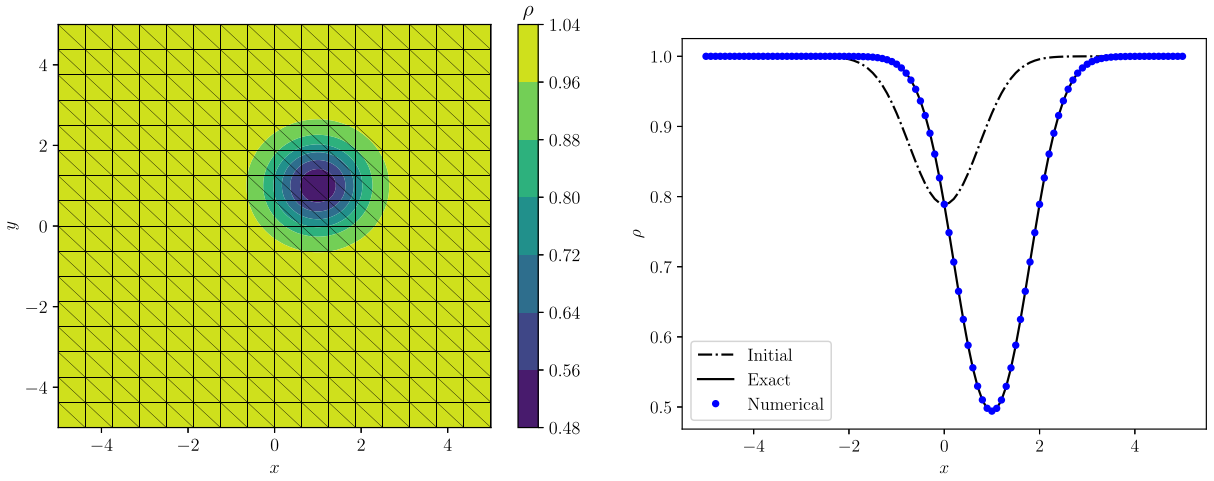
5. Impact

Although DG methods are increasing in popularity, the barrier of entry remains high since they are generally more complicated

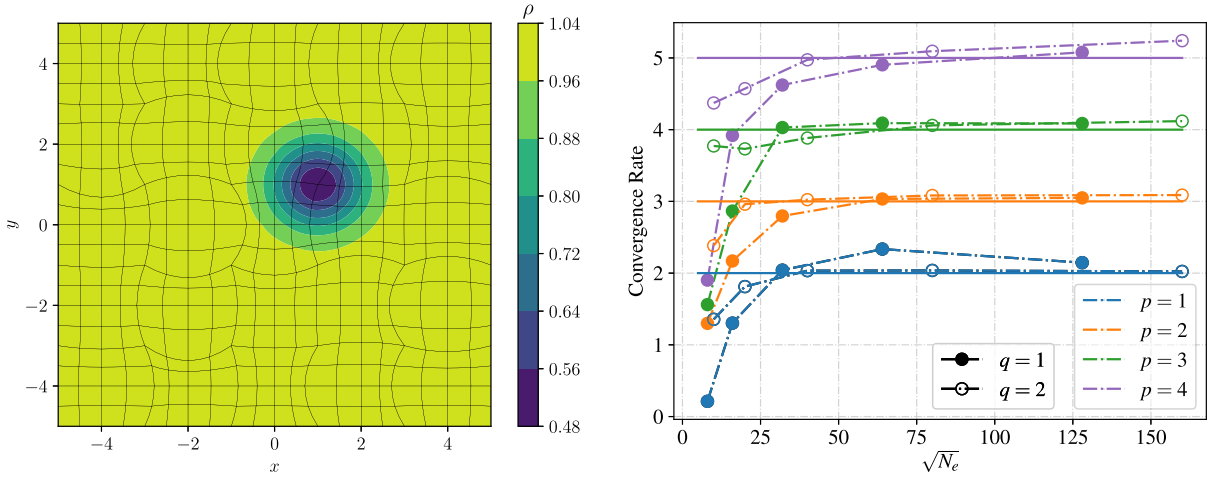


(a) Dissipation relation (b) Dispersion relation

Fig. 3. (a) Dissipation and (b) dispersion relations for $p = 1$ to $p = 7$ with an upwind flux.



(a) Density contours on $q = 1$ triangular mesh (b) Density line slice through $q = 1$ triangular mesh

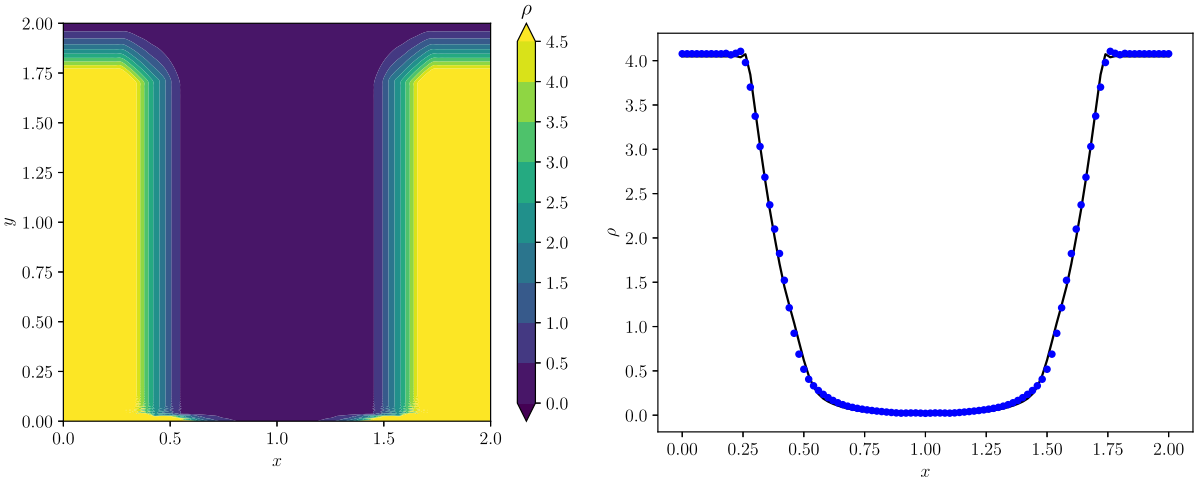


(c) Density contours on $q = 2$ quadrilateral mesh (d) Convergence rates

Fig. 4. Numerical solution for the propagation of an isentropic vortex. (a) Density contours at $t = 1.0$ s on 1024 triangular elements. (b) Line slice at $y = 1$ comparing the initial, exact, and numerical solutions. (c) Variation of convergence rate with $\sqrt{N_e}$ for different polynomial orders based on L_2 density error. The expected convergence rate is represented with solid lines.

Table 1
Current set of features implemented in Quail.

Basis and Geometry			
Shape	Segment	Triangle	Quadrilateral
Nodal Basis	Lagrange	Lagrange	Lagrange
Modal Basis	Legendre	Legendre	H1 Hierarchic [34]
Quadrature Rules	Gauss–Legendre, Gauss–Lobatto	Gauss–Legendre, Dunavant [35]	Gauss–Legendre, Gauss–Lobatto
Stabilization		**Time-steppers**	
Positivity-Preserving Limiter [33] WENO Limiter [36] Artificial Viscosity		Forward Euler RK4 LSRK4 [37]	
Solvers		SSPRK3 [38]	
DG ADERDG [39]		ADER [39] Strang [40]	
Physics		Simpler [41]	
Scalar advection–diffusion Burgers' Equation Euler/Navier–Stokes Equations			



(a) Density contours (b) Density line slice

Fig. 5. Numerical solution for the 2D Riemann problem with a gravity source term. (a) Density contours at $t = 0.6$ s on 25,600 quadrilateral elements. (b) Line slice at $y = 1.7875$ comparing the numerical solution of a 6400 element case (blue symbols) to the 25,600 element case (solid black line). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

than conventional low-order numerical schemes, like finite volume and finite difference methods, and there are comparatively fewer learning resources. To help address this issue, Quail aims to facilitate learning for students, scientists, and engineers eager to enter the field. Key implementation details that are not readily discussed in the literature are made clear with this lightweight, user-friendly code. The simplicity of Quail, contrasted with the complexity of large-scale codebases, makes it more digestible to newcomers who desire hands-on access to DG schemes.

This simplicity, along with modularity and Python as the language of choice, makes Quail also conducive to rapid prototyping. Physical models can be easily incorporated to assess the performance of DG schemes in new contexts. In addition, novel features and methods can be quickly implemented, tested, and applied to model problems before being added to production codes. This can accelerate the process of algorithmic development. The extensive use of vectorized NumPy operations [1] enables simulations of a wide range of 1D and 2D configurations of relevant physical scale and resolution. Links to video tutorials provided in the repository (Code metadata) further illustrate how to use Quail and add certain features, such as boundary conditions, in a straightforward manner.

6. Conclusions

Quail is a lightweight discontinuous Galerkin code written in Python. It is designed for teaching and prototyping without the unwieldy intricacies of production codes. Code clarity, modularity, and ease of use are major focuses. Currently, Quail solves 1D and 2D first- and second-order partial differential equations. The software architecture and functionalities are discussed. Isentropic vortex propagation and a 2D Riemann problem with a gravity source term governed by the Euler equations are used as illustrative examples. Quail can impact the community by reducing the barrier of entry for newcomers and accelerating algorithmic developments. Future work will entail incorporation of new features, such as additional limiters, equation sets, and numerical schemes.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Funding from an Early Career Faculty grant (NNX15AU58G) from the NASA Space Technology Research Grants Program, National Science Foundation (DGE-1656518 and 1909379), and the DoD SMART Scholarship are gratefully acknowledged.

References

[1] Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ, Kern R, Picus M, Hoyer S, van Kerkwijk MH, Brett M, Haldane A, del Río JF, Wiebe M, Peterson P, Gérard-Marchant P, Sheppard K, Reddy T, Weckesser W, Abbasi H, Gohlke C, Oliphant TE. Array programming with NumPy. Nature 2020;585(7825):357–62.

[2] Hunter JD. Matplotlib: A 2D graphics environment. Comput Sci Eng 2007;9(3):90–5.

[3] Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, van der Walt SJ, Brett M, Wilson J, Millman KJ, Mayorov N, Nelson ARJ, Jones E, Kern R, Larson E, Carey CJ, Polat I, Feng Y, Moore EW, VanderPlas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, van Mulbregt P, SciPy 10 Contributors. SciPy 1.0: Fundamental Algorithms for scientific computing in python. Nature Methods 2020;17:261–72.

[4] Cockburn B, Karniadakis GE, Shu CW. The development of discontinuous Galerkin methods. In: Cockburn B, Karniadakis GE, Shu CW, editors. Discontinuous Galerkin Methods. Springer, Berlin, Heidelberg; 2000, p. 3–50.

[5] Hesthaven JS, Warburton T. Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, And Applications. Springer; 2007.

[6] Hartmann R, Held J, Leicht T, Prill F. Discontinuous Galerkin methods for computational aerodynamics – 3D adaptive flow simulation with the DLR PADGE code. Aerosp Sci Technol 2010;14(7):512–9.

[7] Brazell MJ, Mavriplis DJ, Yang Z. Mesh-resolved airfoil simulations using finite volume and discontinuous Galerkin solvers. AIAA J 2016;54(9):2659–70.

[8] Zwick D, Balachandar S. Dynamics of rapidly depressurized multiphase shock tubes. J Fluid Mech 2019;880:441–77.

[9] Ching EJ, Brill SR, Barnhardt M, Ihme M. A two-way-coupled Euler-Lagrange method for simulating multiphase flows with discontinuous Galerkin schemes on arbitrary curved elements. J Comput Phys 2020;405.

[10] Jacobs G, Hesthaven JS. High-order nodal discontinuous Galerkin particle-in-cell method on unstructured grids. J Comput Phys 2006;214:96–121.

[11] Pfeiffer M, Hindenlang F, Binder T, Copplestone SM, Munz C-D, Fasoulas S. A particle-in-cell solver based on a high-order hybridizable discontinuous Galerkin spectral element method on unstructured curved meshes. Comput Methods Appl Mech Eng 2019;349:149–66.

[12] Chu R, Endeve E, Hauck CD, Mezzacappa A. Realizability-preserving DG-IMEX method for the two-moment model of fermion transport. J Comput Phys 2019;389:62–93.

[13] Teukolsky SA. Formulation of discontinuous Galerkin methods for relativistic astrophysics. J Comput Phys 2016;312:333–56.

[14] Kabaria H, Lew AJ, Cockburn B. A hybridizable discontinuous Galerkin formulation for non-linear elasticity. Comput Methods Appl Mech Eng 2015;283:303–29.

[15] Nguyen VP. Discontinuous Galerkin/extrinsic cohesive zone modeling: Implementation caveats and applications in computational fracture mechanics. Eng Fract Mech 2014;128:37–68.

[16] Huynh HT. A flux reconstruction approach to high-order schemes including discontinuous Galerkin methods. In: 18th AIAA Computational Fluid Dynamics Conference. AIAA 2007-4079; 2007.

[17] Liu Y, Vinokur M, Wang ZJ. Spectral difference method for unstructured grids I: Basic formulation. J Comput Phys 2006;216(2):780–801.

[18] Wang ZJ, Liu Y, May G, Jameson A. Spectral difference method for unstructured grids II: extension to the Euler equations. J Sci Comput 2007;32(1):45–71.

[19] Wang ZJ, Fidkowski K, Abgrall R, Bassi F, Caraeni D, Cary A, Deconinck H, Hartmann R, Hillewaert K, Huynh H, Kroll N, May G, Persson P-O, van Leer B, Visbal M. High-order CFD methods: Current status and perspective. Int J Numer Methods Fluids 2013;72:811–45.

[20] Riviere B. Discontinuous Galerkin Methods For Solving Elliptic And Parabolic Equations: Theory And Implementation. SIAM; 2008.

[21] Cantwell CD, Moxey D, Comerford A, Bolis A, Rocco G, Mengaldo G, De Grazia D, Yakovlev S, Lombard J-E, Ekelschot D, Jordi B, Xu H, Mohamied Y, Eskilsson C, Nelson B, Vos P, Biotto C, Kirby RM, Sherwin SJ. Nektar++: An open-source spectral/hp element framework. Comput Phys Comm 2015;192:205–19.

[22] Hindenlang F, Gassner GJ, Altmann C, Beck A, Staudenmaier M, Munz C-D. Explicit discontinuous Galerkin methods for unsteady problems. Comput & Fluids 2012;61:86–93.

[23] Alnæs M, Blechta J, Hake J, Johansson A, Kehlet B, Logg A, Richardson C, Ring J, Rognes ME, Wells GN. The FEniCS project version 1.5. Arch Numer Softw 2015;3(100).

[24] Witherden FD, Farrington AM, Vincent PE. PyFR: AN open source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach. Comput Phys Comm 2014;185(11):3028–40.

[25] Arndt D, Bangerth W, Davydov D, Heister T, Heltai L, Kronbichler M, Maier M, Pelteret J-P, Turcksin B, Wells D. The deal.II finite element library: Design, features, and insights. Comput Math Appl 2021;81:407–22.

[26] Klöckner A, Warburton T, Hesthaven JS. Solving wave equations on unstructured geometries. In: GPU Computing Gems Jade Edition. Elsevier; 2012, p. 225–42.

[27] Geuzaine C, Remacle JF. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. Int J Numer Methods Eng 2009;79(11):1309–31.

[28] Dumbser M, Zanotti O. Very high order $P_N P_M$ schemes on unstructured meshes for the resistive relativistic MHD equations. J Comput Phys 2009;228:6991–7006.

[29] Bornhoft BJ, Ching EJ, Ihme M. Time integration considerations for the solution of reacting flows using discontinuous Galerkin methods. In: AIAA Scitech 2021 Forum. AIAA 2021-0745; 2021.

[30] Wang C. Reconstructed discontinous Galerkin method for the compressible Navier-Stokes equations in arbitrary Langrangian and Eulerian formulation. (Ph.D. Thesis), North Carolina State University; 2017.

[31] Yu M, Wang Z, Liu Y. On the accuracy and efficiency of discontinuous Galerkin, spectral difference and correction procedure via reconstruction methods. J Comput Phys 2014;259:70–95.

[32] Zhang X, Shu C-W. Positivity-preserving high order discontinuous Galerkin schemes for compressible Euler equations with source terms. J Comput Phys 2011;230(4):1238–48.

[33] Zhang X, Shu C-W. On positivity-preserving high order discontinuous Galerkin schemes for compressible Euler equations on rectangular meshes. J Comput Phys 2010;229(23):8918–34.

[34] Solin P, Segeth K, Dolezel I. Higher-Order Finite Element Methods. CRC; 2003, p. 55–60.

[35] Dunavant D. High degree efficient symmetrical Gaussian quadrature rules for the triangle. Int J Numer Methods Eng 1985;21:1129–48.

[36] Zhong X, Shu C-W. A simple weighted essentially nonoscillatory limiter for Runge–Kutta discontinuous Galerkin methods. J Comput Phys 2013;232(1):397–415.

[37] Carpenter MH, Kennedy C. Fourth-order 2N-storage Runge-Kutta schemes. NASA Langley Research Center, NASA Report TM 109112, Richmond, VA, USA; 1994.

[38] Spiteri R, Ruuth S. A new class of optimal high-order strong-stability-preserving time discretization methods. SIAM J Numer Anal 2002;40(2):469–91.

[39] Dumbser M, Enaux C, Toro E. Finite volume schemes of very high order of accuracy for stiff hyperbolic balance laws. J Comput Phys 2008;227:3971–4001.

[40] Strang G. On the construction and comparison of difference schemes. SIAM J Numer Anal 1968;5(3).

[41] Wu H, Ma P, Ihme M. Efficient time-stepping techniques for simulating turbulent reactive flows with stiff chemistry. Comput Phys Comm 2019;243:81–96.