An implicit, discontinuous Galerkin Chimera solver using automatic differentiation

Nathan A. Wukie*, Paul D. Orkwis†, and Mark G. Turner‡
University of Cincinnati, Cincinnati, Ohio, 45221

The development of a fully-implicit, discontinuous Galerkin solver based on Chimera, overset grids is presented. Newton's method is used as a nonlinear solver for governing equation sets, enabled by an intrinsic automatic differentiation capability that computes the linearization of the spatial scheme. A GMRES matrix-solver is used to solve the linear system along with a block-ILU0 preconditioner. Results include solutions of the Euler equations for subsonic flow over a smooth bump as well as over a circular cylinder. The convergence rates for spatial orders of accuracy for the solver match the analytical rates. Up to 4th-order accuracy is shown for 3rd-order polynomials (P3). Additionally, quadratic convergence is demonstrated for the nonlinear solver; verifying that the automatic differentiation capability was implemented successfully. Solutions converged in 7 or fewer iterations. Results are presented utilizing the Chimera overset grid capability, demonstrating that quadratic convergence is maintained with the Chimera grid interfaces.

Nomenclature

Q	Solution variable
R	Residual of governing equations
$\vec{F^a}$	Advective flux vector
$\vec{F^d}$	Diffusive flux vector
S	Source function
$\vec{c}$	Convection velocity vector
$\vec{n}$	Normal vector
t	Time
ψ	Legendre basis polynomial
ξ, η, ζ	Coordinates in a reference element
P	Order of 1D polynomial expansion
M_∞	Freestream Mach number

Subscript

i, j, k	Tensor indices

I. Introduction

The discontinuous Galerkin (DG) method for solving partial differential equations has been applied in many fields including acoustics, numerical weather prediction, and aerodynamics.[1–3] Part of the strength of the method exists in its ability to provide arbitrarily high-order accuracy, while maintaining a local computational stencil. It also benefits from taking advantage of a great body of work from the Finite Volume community in its use of numerical fluxes to connect the solution between discontinuous elements.

*Phd Student, Dept. of Aerospace Engineering, ML 70, Cincinnati, Ohio 45221, and AIAA Student Member.
†Bradley Jones Professor, Dept. of Aerospace Engineering, ML 70, Cincinnati, Ohio 45221, AIAA Associate Fellow.
‡Associate Professor, Dept. of Aerospace Engineering, ML 70, Cincinnati, Ohio 45221, AIAA Associate Fellow.

I.A. DG Chimera

The DG method is often used with unstructured computational grids because it is a local scheme. However, the properties of DG are also very attractive for Chimera overset grids. Galbraith et al.[4] developed an implicit, DG-Chimera solver which highlighted the benefits of DG applied to Chimera overset grids. This included the elimination of orphan nodes and fringe points that traditionally exist in Finite Volume-based Chimera solvers. Additionally, the ability of DG to accommodate curved geometry solved a Finite Volume issue with overlapping grids, where a boundary from one grid can exist inside the computational domain of the neighbor grid. This makes the Chimera-DG method quite attractive as a framework for a design tool, as it brings together the desired traits of a high-order accurate solution with straight-forward geometry representation and modification.

I.B. Automatic differentiation

This work targets an implicit formulation that uses Newton's method to solve the governing equations. The implicit Newton solver requires a linearization of the governing equations. A direct formula for the linearization can be obtained by-hand or by using symbolic manipulation tools[5] to simplify the process. Galbraith[6] provides an extensive discussion of the process for deriving and constructing the linearization in the context of a discontinuous Galerkin framework. There are a few unfortunate side-effects of relying on an explicit formula and implementation of the linearization that are also detailed by Galbraith et al.[7] These are reiterated here to highlight the case for automatic differentiation.

- Time investment in deriving the linearization
- Time investment in verifying the implementation
- Aversion to future development efforts due to the difficulty of the derivation/implementation/verification process

If a code is being developed for a single, specific purpose, then it makes sense to invest the time upfront to derive the linearization, implement it, and then verify correctness. If however, there is a desire to frequently test new capabilities and numerical methods, an integrated capability for obtaining the linearization of new algorithms is greatly beneficial.

Automatic differentiation (AD) of numerical algorithms exists in a number of different forms and in many different software tools.[8] Some of these include ADOL-C, ADIFOR, and Tapenade.[9–11] A good overview of the field is given by Bischof et al.[12] Xia et al.[13] applied the Tapenade source transformation tool to linearize a reconstructed discontinuous Galerkin solver for compressible flows on 3D hybrid grids. The Tapenade tool requires that the source code be preprocessed to generate a differentiated version of a procedure. Their work was successful in avoiding an analytical derivation of the linearization. They also reduced the memory requirement of the implicit algorithm by using a truncated solution expansion in computing the linearization. However, the reduced memory technique eliminates quadratic convergence of the Newton solution method.

The approach taken here utilizes DNAD (Dual-Number Automatic Differentiation), which is a data-type defined in Fortran that enables automatic differentiation via operator-overloading.[14,15] This operator overloading technique has been used previously in other tools. One example is the *derivify.h* and *complexify.h* types detailed by Martins et al.,[16] which use operator overloading to implement dual-number and complex-step based AD respectively. The *derivify.h* approach was advocated by Galbraith et al.[7] in combination with C++ expression templates to improve the performance of the class for dynamically allocated arrays of derivatives. They noted that a traditional implementation of dynamically allocated derivative arrays incurs some very significant decreases in performance when computing relatively few derivative terms. They then implemented the class using C++ expression templates to facilitate lazy evaluation of the operations and were able to recouperate a significant percentage of the original performance.

I.C. Present work

The present work joins the concepts of an implicit DG-Chimera scheme with the additional flexibility of operator-overloaded automatic differentiation to linearize the algorithm. This enables a high-order accurate scheme that is well-suited for design studies and geometry modification, in addition to providing a platform for rapidly testing new numerical techniques.

The choice to use DNAD for this work was driven by the desire to embed an intrinsic automatic differentiation capability into the solver framework. This eliminates the traditional requirement of the developer to derive and implement the linearization. It also eliminates dependency on external tools that require preprocessing or that require other in-source designations to facilitate the differentiation. In this way, developers are not required to learn or depend on a separate tool in order to implement a new capability. The DNAD data-type provides an analogous capability to the *derivify.h* Surreal class. For this work, the DNAD tool was modified to enable dynamically allocated arrays of derivatives. One aspect of this work investigates the performance of the static and dynamic derivative arrays as the array sizes increase. It is shown that as the number of derivative components increases, the static and dynamic implementations execute in the same amount of time and the allocation overhead in temporary variables becomes negligible. This has implications for implementation strategies for the discontinuous Galerkin method, where the number of degrees of freedom per element, and corresponding arrays of partial derivatives, can be quite large.

II. Framework development

This section details the different aspects of the framework development, including the spatial discretization, the nonlinear solution method, the automatic differentiation technique, Chimera interfaces, and the governing equations used in the present work. Fortran has been chosen as the primary development language for this effort, as the modern standard supports software concepts of derived and polymorphic types while its primary purpose remains numerical computation.

II.A. The discontinuous Galerkin discretization

The developed framework takes its inspiration from Galbraith[6] and is being designed to handle equation sets in weak conservation form as

$$\frac{\partial Q}{\partial t} + \nabla \cdot \vec{F}(Q, \nabla Q) + S(Q, \nabla Q) = 0 \tag{1}$$

The solution variables are expressed as an expansion in basis polynomials constructed as a tensor product of 1D Legendre polynomials as

$$Q(t, x, y, z) = \sum_{i=0}^{P} \sum_{j=0}^{P} \sum_{k=0}^{P} Q_{ijk}(t) \psi_i(x) \psi_j(y) \psi_k(z) \tag{2}$$

Multiplying by a column of test functions $\boldsymbol{\psi}$ and applying Gauss' Divergence Theorem gives our working form of the discontinuous Galerkin method as

$$\int_{\Omega_e} \boldsymbol{\psi} \frac{\partial Q}{\partial t} d\Omega + \int_{\Gamma_e} \boldsymbol{\psi} \vec{F} \cdot \vec{n} d\Gamma - \int_{\Omega_e} \nabla \boldsymbol{\psi} \cdot \vec{F} d\Omega + \int_{\Omega_e} \boldsymbol{\psi} \vec{S} d\Omega = 0 \tag{3}$$

Numerical Gauss-Legendre quadrature is used to compute the spatial integrals in Equation 3. $3/2(P+1)$ quadrature points per direction are used to evaluate the integrals in order to account for potential nonlinearities in the flux and source terms, as recommended by Kirby and Karniadakis.[17]

II.B. Nonlinear solver

The goal for solving sets of equations in the framework is to use Newton's method, which provides quadratic convergence for solving systems of nonlinear equations. In this case, we are interested in finding solutions such that the spatial residual is zero as

$$R(Q) = 0 \tag{4}$$

The formulation for Newton's method is

$$\frac{\partial R(Q^n)}{\partial Q} \Delta Q^n = -R(Q^n) \tag{5}$$

where $R(Q)$ is the spatial residual from Equation 3 and $\frac{\partial R(Q)}{\partial Q}$ is the matrix containing the linearization of the spatial scheme. Solving for ΔQ, the solution is then updated as

$$Q^{n+1} = Q^n + \Delta Q \tag{6}$$

Newton's method is particularly desirable since the DG method has a high number of degrees of freedom in each element, which puts extra importance on having an efficient method for solving the governing equations.

One challenge associated with Newton's method is that it relies on being relatively 'close' to the solution. In that case, Newton's method converges quadratically. However, in the case that the initial solution is not sufficiently close to the final solution, Newton's method can diverge. A Quasi-Newton method can be used, where a pseudo-time derivative is added to limit the size of the Newton steps until the numerical solution is closer to the final answer. The Quasi-Newton formulation is given as

$$\int_{\Omega_e} \psi \frac{\Delta Q^n}{\Delta \tau_e} d\Omega + \frac{\partial R(Q^n)}{\partial Q} \Delta Q^n = -R(Q^n) \tag{7}$$

This adds a Mass matrix (scaled by the pseudo-time step) to the block-diagonal components of the linearization matrix, $\frac{\partial R(Q)}{\partial Q}$. The pseudo-time step is computed as

$$d\tau = \frac{CFL^n h_e}{\bar{\lambda}_e} \tag{8}$$

where $h_e = \sqrt[3]{\Omega_e}$, and $\bar{\lambda}_e = |\bar{V}_e| + \bar{c}$ is the mean characteristic speed. The CFL term is computed from the ratio of the initial and current residual norms as

$$CFL^n = CFL^0 \frac{||R(Q^0)||_2}{||R(Q^n)||_2} \tag{9}$$

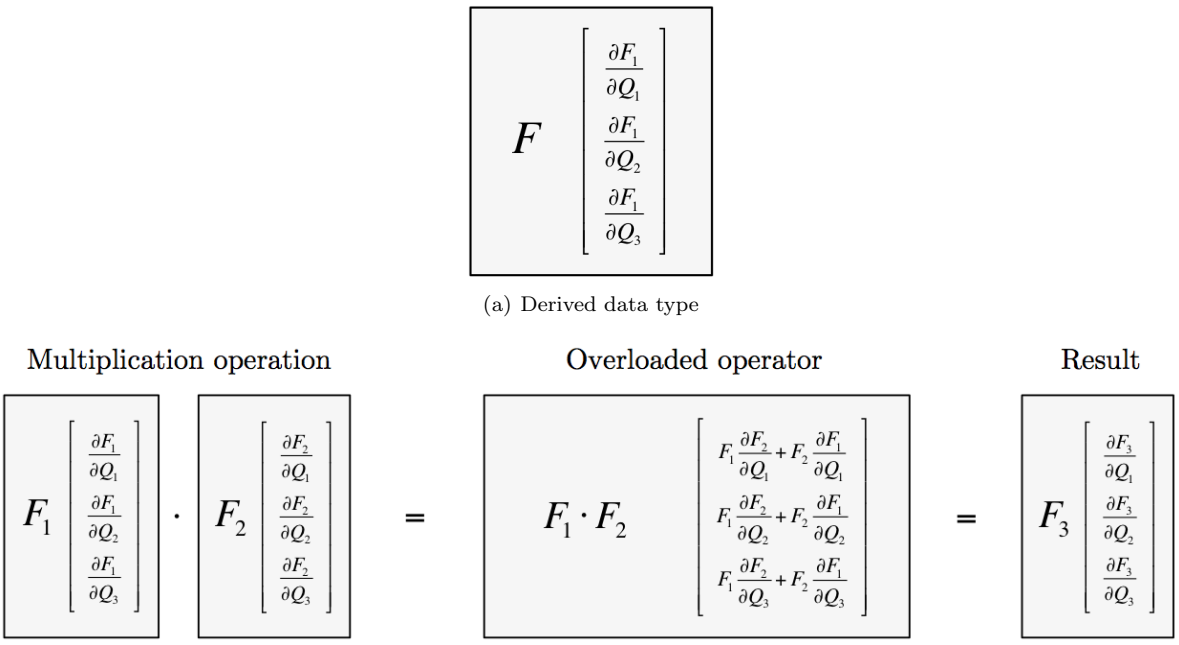
This approach was originally proposed by Orkwis and McRae.[5] In this way, as the spatial residual converges to zero, the time-step increases to infinity and the original Newton's method is recovered.

II.C. Automatic differentiation

Newton's method requires the linearization of the spatial scheme, $\frac{\partial R(Q)}{\partial Q}$. As mentioned previously, this can be computed by deriving analytical expressions for the terms contributing to the linearization and implementing them into the framework. That method tends to be error-prone, tedious, and time-consuming, particularly if one wishes to have a flexible framework for implementing and testing new methods; a new flux scheme or boundary condition for example.

The approach taken in this work takes advantage of concepts from computer science and differential calculus to obtain a general, automatic differentiation capability. A new data type is defined to contain a value and an array of partial derivatives, diagrammed in Figure 1(a). All operators are then overloaded to compute the value of the operation and also the derivative of the operation with respect to the solution variables via the chain-rule definition of that operation, shown in Figure 1(b). The automatic differentiation file containing the new data-type and overloaded operators used in the present work was developed previously by Spall and Yu[14] and Yu and Blair.[15] Their work is titled DNAD (Dual Number Automatic Differentiation), which is a Fortran implementation. However, implementations exist for other languages.

One method of implementing the automatic differentiation capability is to replace every floating point value in code with the newly defined data-type containing a value and derivatives. The length of the derivative array in each instance is set to the number of solution variables in the simulation. The initial value of the derivatives are all zero, except for the derivative of a solution value with respect to itself, which is one. Once all contributions to $R(Q)$ have been computed, the values for $\frac{\partial R(Q)}{\partial Q}$ can be taken from the arrays of partial derivatives in $R(Q)$. This is diagrammed in Figure 2. A more practical implementation can be achieved by recognizing that for the current scheme, solution variables only depend on their immediate neighbors, so the derivative arrays allocated and computed in a calculation can be significantly reduced. For example, in the context of the discontinuous Galerkin discretization used by the present work, boundary and volume integrals are computed for each element. When a boundary integral is computed, the function value on the boundary depends only on the solution variables in the neighboring elements. So, the derivative arrays only need to be constructed to track the derivatives with respect to the neighboring variables. This reduces compute work



(a) Derived data type

(b) Example differentiation of multiplication operation

Figure 1. Automatic differentiation data type and example diagram.

by ignoring contributions from non-neighboring elements. Some additional computational savings can be achieved by recognizing that some matrices (interpolation matrices) used in the DG method can be declared as standard floating-point values instead of AD-types, since they do not depend on the solution variables.

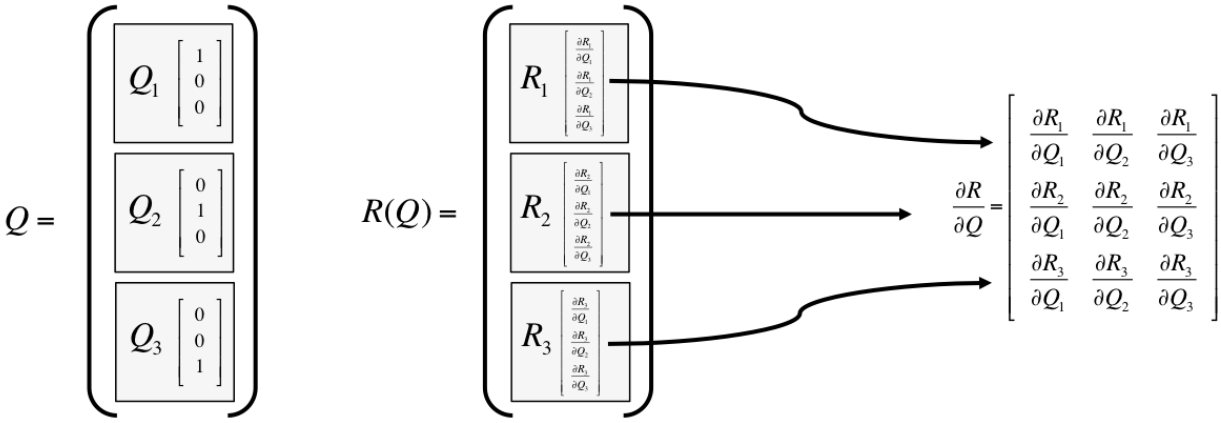


Figure 2. Diagram of automatic differentiation process for obtaining the linearization data

In this way, when a developer implements a new capability into the framework (boundary condition, equation set, etc.) only the contribution to the spatial residual, $R(Q)$, is required. The intrinsic automatic differentiation capability computes the contributions to $\frac{\partial R(Q)}{\partial Q}$ without any additional requirement on the developer. This technique also avoids dependence on external tools for preprocessing and avoids any requirement for in-source designations to inform the differentiation process, greatly simplifying the framework for developers.

II.D. Chimera interfaces

Chimera overset grids are used in the framework to accommodate complex geometry. The approach for inter-grid communication follows Galbraith et al.,[4] where elements are connected only through the boundary

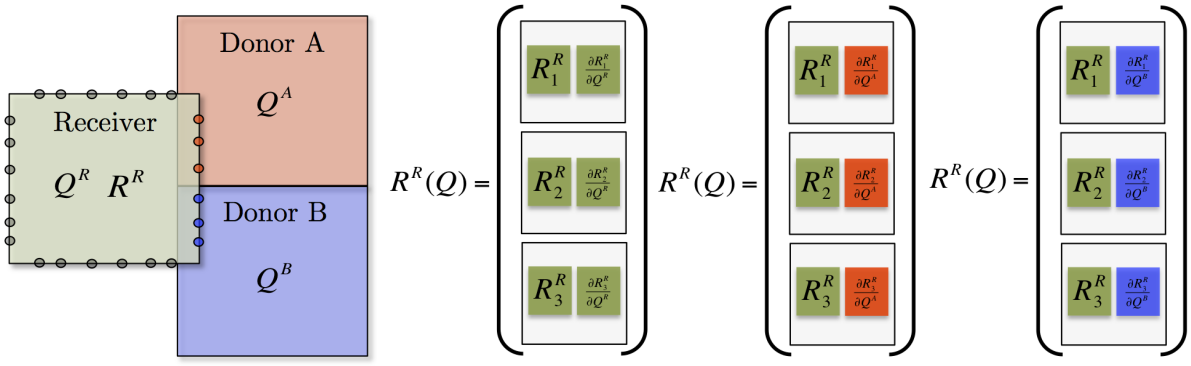
quadrature nodes. This contrasts the work by Wurst et al.,[18] where inter-grid communication occurs across volume quadrature nodes.

II.D.1. Residual evaluation

A diagram of a Chimera interface in the context of a DG discretization is shown in Figure 3(a). Given an element with a Chimera face (Receiver), donor elements must be found such that the quadrature nodes from the receiver are located within a donor element. In order to compute the residual contribution to the receiver element from the Chimera face, the solution variables from the receiver and donor elements must be evaluated at the location of each quadrature node. The solution variables in a given element are defined in a local coordinate system for that element as $Q(\xi, \eta, \zeta)$, where ξ, η, ζ are the computational coordinates defined on a reference element. In order to evaluate the solution variables from a donor element at a receiver quadrature point, the location of the quadrature point must be found in the local coordinate system of the donor element. That is, we must find $(\xi_{Donor}, \eta_{Donor}, \zeta_{Donor}) = f(x_{gq}, y_{gq}, z_{gq})$. This is computed using a method outlined by Galbraith,[6] where Newton's method is used to iteratively solve for the donor local coordinates. Once the local coordinates in the donor element have been found, the solution from the donor element can be computed as an exterior solution value at the quadrature node. With internal and external solution values for the complete set of quadrature nodes, the contribution to the residual of the receiver element can be computed in the same way as the interior scheme.

II.D.2. Linearization of Chimera interfaces

In order to maintain quadratic convergence of the nonlinear Newton solver, the linearization of the residual in the Receiver element with respect to the Donor elements ($\frac{\partial R^R}{\partial Q^A}$, $\frac{\partial R^R}{\partial Q^B}$) must be computed and added to the linearization matrix for Newton's method. In the present work, the linearization of a given element with respect to a neighboring element is facilitated via the intrinsic automatic differentiation capability and is computed by initializing the derivative arrays of the neighboring solution variables, evaluating the modal solution variables at the discrete quadrature nodes, evaluating the residual function at the quadrature nodes, and then projecting back to the original modal function space. This same process is used for the Chimera faces except it is repeated for each element that contributes to the residual function. So, for the case shown in Figure 3(a), three residual evaluations are computed for the Chimera face; one with the *Donor A* variables initialized for differentiation, one for the *Donor B* variables initialized, and one for the *Receiver* variables initialized. This is diagrammed in Figure 3(b). In this way, a complete linearization of the spatial scheme is maintained, which is essential for achieving quadratic convergence with the nonlinear Newton solver.



(a) Chimera interface diagram (b) Multiple residual evaluations for computing linearization of Chimera boundary

Figure 3. Diagrams for Chimera-DG interface

II.E. Code development practices

The framework described here is developed in modern Fortran (F2008), which provides constructs for safe dynamic memory management, polymorphic data types, and modules. The Fortran module capability is an excellent method for separating portions of code into easily recognized units in addition to facilitating

global memory access and automatically generated explicit interfaces. CMake is used to manage build configurations and enable portability of the code. Automated unit and regression testing is used to ensure that consistent behavior of the code is maintained and expected behavior of the code is preserved. The open-source tool pFUnit provides the testing capability, which is implemented in native Fortran. Jenkins is used as a continuous integration server that manages automated checkout, building, and testing of the code on daily and weekly schedules, along with automated testing of each new commit to a central git repository.

II.F. Euler equations

The Euler equations governing inviscid, compressible flows were implemented into the framework. The solution variables and advective flux vectors are given as

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix} \quad \vec{F}(Q) = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho uH \end{bmatrix}, \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ \rho vH \end{bmatrix}, \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ \rho wH \end{bmatrix} \tag{10}$$

where $H = \frac{\rho E + p}{\rho}$ is the total enthalpy. The fluid is assumed to be an ideal gas and the pressure is computed as

$$p = (\gamma - 1)(\rho E - \frac{1}{2}\rho \vec{V}^2) \tag{11}$$

The Roe scheme[19] is used to compute the numerical advective boundary fluxes.

III. Results

Results from this work include solutions of smooth bump, circular cylinder, and airfoil geometries. These geometries include both single-block, and chimera-block domains. Verification studies are completed for the smooth bump and circular cylinder geometries to confirm that the implementation of the framework was successful. 3D smooth bump and airfoil geometries are included to demonstrate a more general analysis capability. For all calculations, $CFL^0 = 0.5$ was used. Also, solutions of higher polynomial order are initialized from the next lower-order solution.

III.A. Verification

Two metrics were used in this study to verify the implementation. First, the spatial orders of accuracy for the numerical scheme are investigated. For the Euler equations at a subsonic condition everywhere in the domain, the flow is isentropic. The entropy generation in the numerical solution is then used as an error metric for verifying the spatial order of accuracy. It is computed as

$$EntropyError = \sqrt{\frac{\int \left(\frac{\frac{p}{\rho^\gamma} - \frac{p_\infty}{\rho_\infty^\gamma}}{\frac{p_\infty}{\rho_\infty^\gamma}} \right)^2 d\Omega}{\int d\Omega}} \tag{12}$$

A series of computational grids of increasing refinement are generated for each problem. The entropy error is then computed from the converged solution on each computational grid and the entropy error is plotted against the grid resolution parameter, h_s.

$$h_s = \sqrt{\frac{1}{DOF_{Total}}} \tag{13}$$

This is defined to account for the total number of degrees of freedom in the domain.

The second metric for verifying the implementation was the nonlinear convergence of the implicit Newton solver. This is investigated to confirm that the linearization of the algorithm, via the automatic differentiation technique, is being computed correctly. Newton's method is known to be sensitive to the accuracy of the

linearization. If there are errors in the derivation or implementation, the nonlinear solver may converge, but the convergence rate will be linear or superlinear instead of the desired quadratic convergence rate. If the convergence rate of the nonlinear solver is observed to be quadratic, this is a verification that the automatic differentiation technique was implemented successfully and producing the expected data for the linearization.

III.B. 2D smooth bump

The first test case is a smooth bump geometry. The bump is defined by the following Gaussian profile

$$y(x) = 0.0625e^{-25x^2} \tag{14}$$

The domain is represented by a 6x2x2 mesh using quartic elements. Total pressure and total temperature were specified at the inlet boundary. Constant static pressure was specified at the outlet boundary. The pressure ratio corresponds to a freestream Mach number of $M = 0.5$. First, third, fifth, and seventh-order accurate solutions were computed and are shown in Figure 4 demonstrating the higher-order accurate solution capability as well as the ability of the method to accommodate curved geometry. In the results presented here, P is the order of the 1D polynomials used to construct the solution expansion. For example, P2 designates a quadratic expansion, for which third-order accuracy is expected.

The verification metrics for the smooth bump case are shown in Figure 5. Figure 5(a) shows the achieved spatial orders of accuracy overlayed against the analytical rates of convergence. Excellent agreement between the numerical and analytical rates of convergence is demonstrated, except for the first-order accurate (P0) calculations. This is likely due to additional errors introduced by the difference in the element and solution expansions. Additionally, quadratic convergence of the nonlinear solver is demonstrated in Figure 5(b). Fully-converged solutions are obtained in 6-8 Quasi-Newton steps.

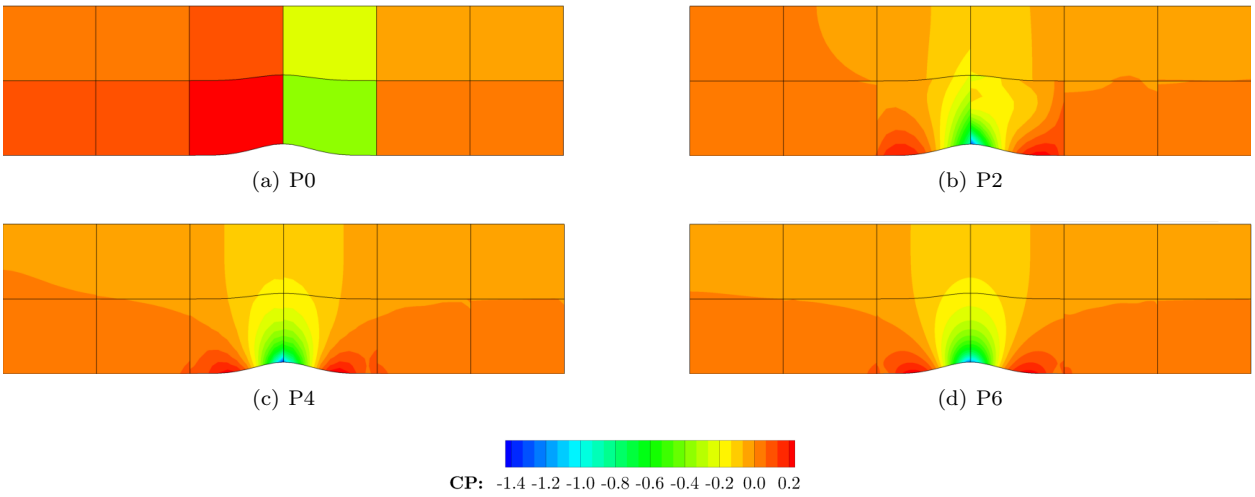
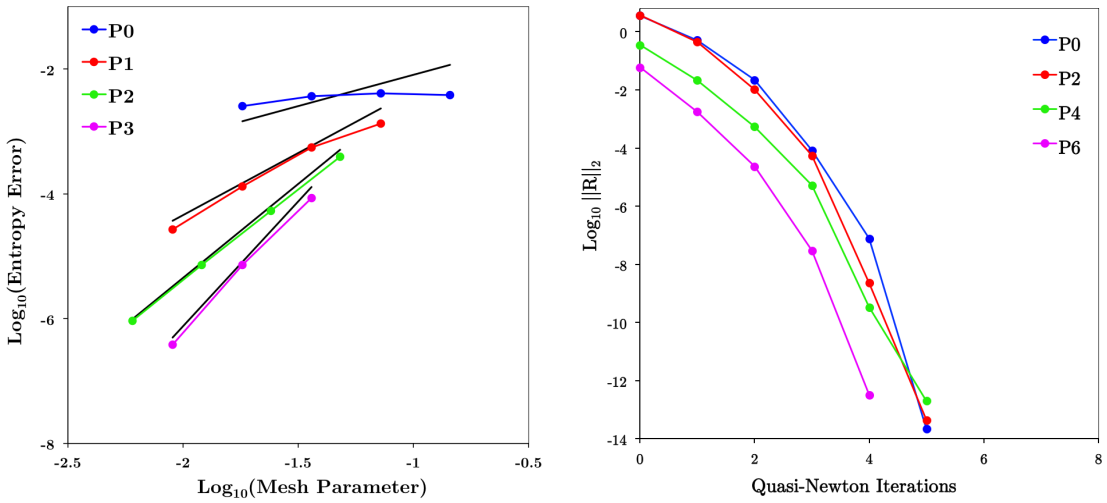


Figure 4. 2D, Smooth Bump solutions of pressure coefficient, Cp. 6x2x2 quartic-element mesh.

III.C. Circular cylinder

The second verification case consists of inviscid flow over a circular cylinder. The domain is represented by four block-structured grids that overlap along the diagonals of the domain, demonstrating the Chimera overset capability. An expanded view of the Chimera grid for this test case is shown in Figure 6. Total pressure and total temperature were specified at the inlet boundary. Constant static pressure was specified at the outlet boundary. The pressure ratio corresponds to a freestream Mach number of $M = 0.2$. Solutions are shown in Figure 7. It is shown that as the solution expansion order is increased, the solution converges to nearly symmetric contours of pressure coefficient. The importance of using higher-order element mappings to accommodate the curvature of solid boundaries was highlighted by Bassi and Rebay,[20] where they noted that using traditional linear element mappings produced low-quality, unsymmetric solutions; even on refined grids. Producing the nearly symmetric contours shown in Figure 7(d) gives additional confidence that the spatial scheme has been implemented correctly.



(a) Spatial orders of accuracy.

(b) Nonlinear convergence history.

Figure 5. Smooth Bump verification

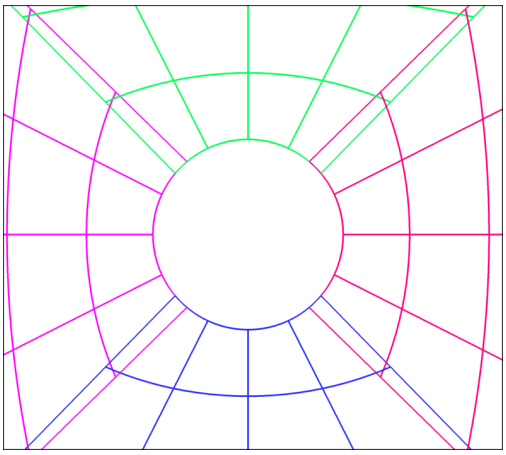


Figure 6. Expanded view of Chimera grid for circular cylinder

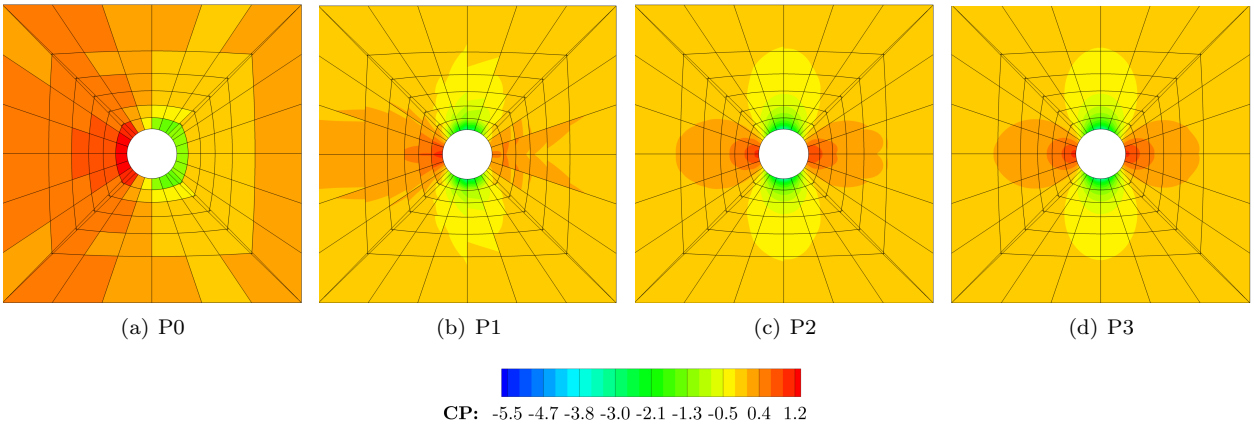
Verification metrics for the circular cylinder case are shown in Figure 8. The spatial orders of accuracy agree well with the analytical rates. Quadratic convergence of the nonlinear solver is still observed as solutions are obtained in 8-9 Quasi-Newton steps. These successful verification metrics confirm that spatial orders of accuracy along with Quadratic convergence are preserved when using the Chimera interfaces.

III.D. 3D smooth bump

The framework was set up as a three-dimensional code. '2D' simulations are performed on 3D elements; using a single element in the third dimension. To demonstrate the 3D capability, a 3D version of the smooth bump test case was constructed using the profile

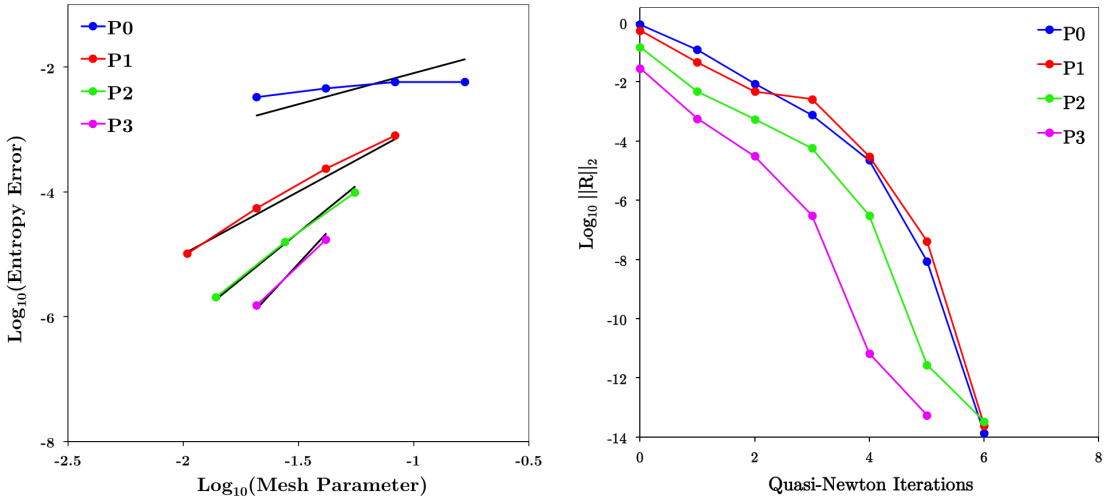
$$y(x,z) = 0.0625e^{-25x^2}e^{-25z^2} \tag{15}$$

Figure 9(a) shows the 3D bump geometry. A total pressure and temperature inlet boundary condition was used along with a constant static pressure outlet boundary condition. The pressure ratio for the flow corresponded to a freestream Mach number of 0.5. Figures 9(b) and 9(c) show results for the coefficient of pressure on the surface geometry and also as isosurfaces, demonstrating the 3D flow features.



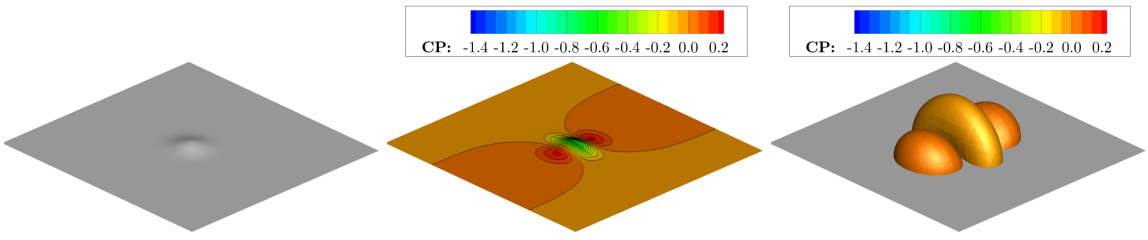
(a) P0 (b) P1 (c) P2 (d) P3

Figure 7. Circular Cylinder (Chimera grid) solutions of pressure coefficient, Cp.



(a) Spatial orders of accuracy. (b) Nonlinear convergence history.

Figure 8. Circular Cylinder (Chimera grid) - Verification.



(a) 3D Smoothbump Geometry (b) Surface pressure coefficient, Cp (c) Isosurfaces of pressure coefficient

Figure 9. 3D Smooth Bump : $M_\infty = 0.5$, P2

III.E. NACA 2412 airfoil

A NACA 2412 airfoil was chosen to demonstrate a more practical simulation capability. Simulations for $\alpha = 0.0$ and $\alpha = 4.0$ angles of attack were run at a freestream Mach number of $M = 0.2$. A quartic-element, O-grid mesh was generated for the airfoil component using a blunt trailing edge. This was set in a coarser, linear background mesh. These are shown in Figures 10(a) and 10(b) for the $\alpha = 0.0$ and $\alpha = 4.0$ settings. A total pressure, total temperature inlet boundary condition was used along with a static pressure outlet boundary condition. Slip-wall boundary conditions were applied at the top and bottom boundaries as well as along the airfoil geometry. Contours of pressure coefficient can be seen in Figures 10(c) and 10(d). The airfoil surface pressure coefficient data was extracted for both the $\alpha = 0.0$ and $\alpha = 4.0$ configurations and compared against predictions from the airfoil design tool, Xfoil.[21] The comparison of pressure coefficient data is shown in Figures 10(e) and 10(f). The data from the present work compares well with the Xfoil prediction. Some discrepancies can be seen around the trailing edge and this is likely due to inadequate grid resolution.

III.F. Performance of Static/Dynamic AD

In the present work, the automatic differentiation data-type and operators from the DNAD tool were extended to allow dynamically allocated arrays of derivatives. The original and modified types are shown in Code blocks 1 and 2. Galbraith et al.[7] investigated the performance of the static and dynamic data types for small derivative array sizes. They found that a traditional implementation of the dynamic type incurred significant computational overhead. They recovered computational savings by implementing the automatic differentiation type using C++ expression templates to delay evaluation of operators, avoiding the creation of temporary variables. Their demonstration used relatively few derivatives.

Code 1. Statically declared AD type

```
type, public :: dual_num
    real(dbl_ad)                :: x_ad_           ! function value
    real(dbl_ad)                :: xp_ad_(NDV_AD)  ! derivatives
end type dual_num
```

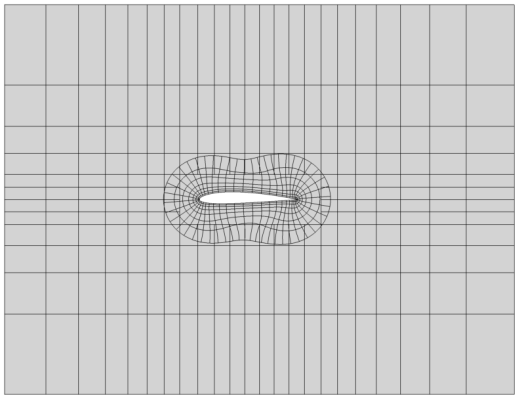
Code 2. Dynamically declared AD type

```
type, public :: dual_num_d
    real(dbl_ad)                :: x_ad_           ! function value
    real(dbl_ad), allocatable   :: xp_ad_(:)       ! derivatives
end type dual_num_d
```

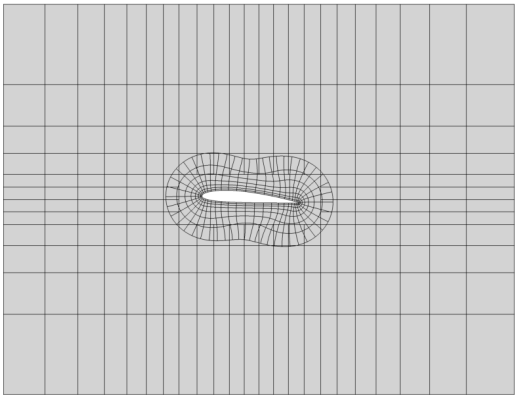
In the context of a discontinuous Galerkin discretization, the derivative arrays can be quite large, due to the large number of degrees of freedom that exist to represent the solution variables. The performance of the static and dynamic AD types was of interest in the case of large derivative arrays. A simple test case was constructed that highlights a performance issue with the dynamically allocated type. Particularly, the need for temporary variables and the allocation overhead that occurs in the overloaded operators for the temporary derivative arrays. This was investigated by performing a simple addition operation as

$$a = b + c \tag{16}$$

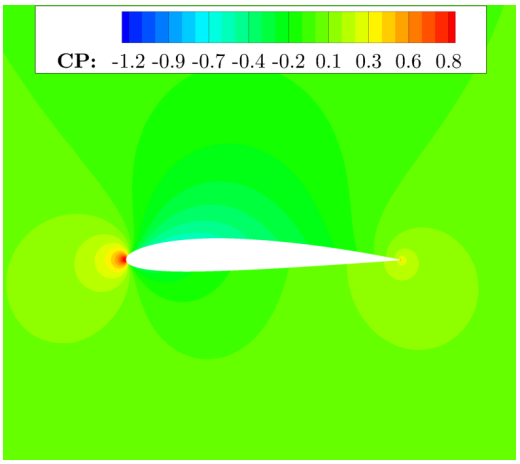
where a, b and c are declared as AD types. This was repeated in a loop 10^7 times. The Fortran code was compiled with the O3 optimization flag. The results shown in Figure 11 show the execution time of the addition loop for the AD types with statically and dynamically allocated arrays of derivatives as the array size increases. This result shows that once the number of derivatives being computed is large enough, the time required to allocate memory for a temporary variable no longer dominates the computation. The implication of this for a DG code is that since the number of derivatives being computed is generally quite high, the use of dynamically allocated types can be used without incurring significant overhead allocating temporary variables, while maintaining a greater level of flexibility within the solver.



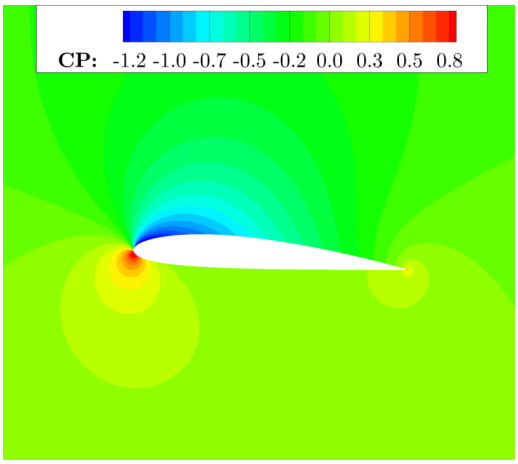
(a) Chimera grid : $\alpha = 0.0$



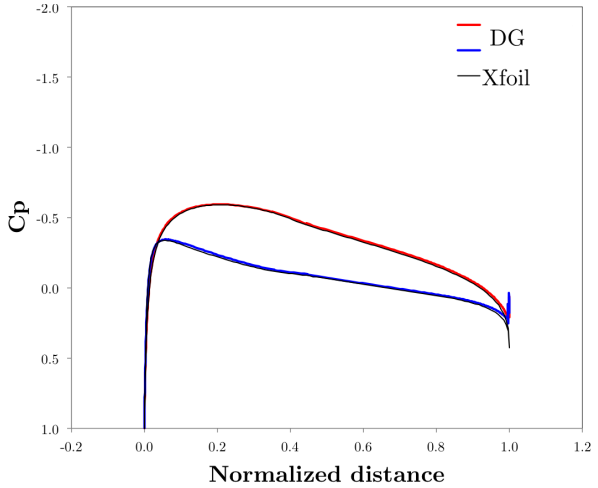
(b) Chimera grid : $\alpha = 4.0$



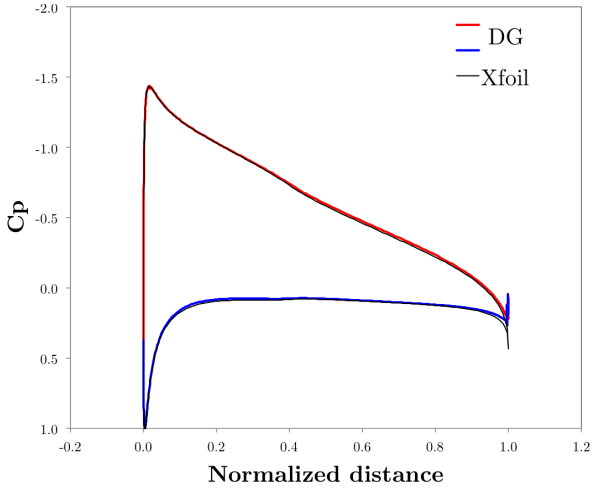
(c) Pressure coefficient (Cp) contours : $\alpha = 0.0$



(d) Pressure coefficient (Cp) contours : $\alpha = 4.0$



(e) Upper/Lower airfoil surface pressure coefficient : $\alpha = 0.0$



(f) Upper/Lower airfoil surface pressure coefficient : $\alpha = 4.0$

Figure 10. NACA 2412 simulation : $M_\infty = 0.2$, P3

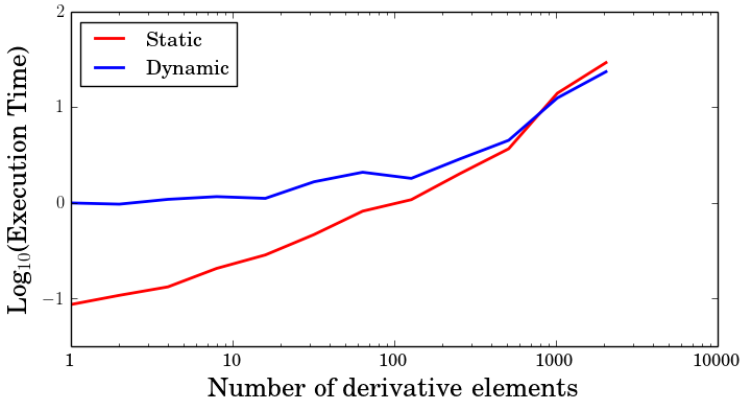


Figure 11. Performance of Addition operator for AD type

IV. Conclusions

A framework has been developed for a fully implicit, discontinuous Galerkin Chimera solver using an automatic differentiation technique for computing the linearization of the spatial scheme. The automatic differentiation capability eliminates traditional work required of developers to derive and implement expressions for the linearization of the spatial scheme. A GMRES matrix solver and block-ILU0 preconditioner are used for solving the linear system of equations. Results from Smooth Bump and Circular Cylinder test cases for the Euler equations demonstrate that the spatial order of accuracy rates match the analytical rates. Up to fourth-order accuracy is shown for third-order polynomials (P3). Additionally, quadratic convergence is demonstrated for the nonlinear solver; verifying that the automatic differentiation capability was implemented successfully. Solutions converge in 7 or fewer iterations. A 3D Smooth Bump and NACA 2412 Airfoil were simulated as well to demonstrate a more general analysis capability. Coefficient of pressure data from the airfoil upper and lower surfaces showed very good comparison against predictions from Xfoil.

The performance of static and dynamic data types associated with the automatic differentiation capability were investigated. It was seen that when small numbers of derivatives are being computed, the dynamically defined data-type is around one order of magnitude more expensive than the statically defined data-type. However, when large numbers of derivatives are being computed, the overhead associated with allocating temporary variables no longer dominates the calculation and the performance of the static and dynamic data types is relatively equal. This indicates that the dynamic type can be used to enable greater flexibility in the code without incurring extra computational overhead.

Current and future efforts include nonreflecting boundary conditions, extension for Navier-Stokes equations, parallelization, and shock capturing.

Acknowledgements

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 1610397.

References

[1]Bauer, M., Dierke, J., and Ewert, R., "Application of a Discontinuous Galerkin Method to Discretize Acoustic Perturbation Equations," *AIAA Journal*, Vol. 49, No. 5, 2011, pp. 898–908.

[2]Kelly, J. F. and Giraldo, F. X., "Continuous and discontinuous Galerkin methods for a scalable three-dimensional nonhydrostatic atmospheric model: Limited-area mode," *Journal of Computational Physics*, Vol. 231, 2012, pp. 7988–8008.

[3]Hartmann, R., Held, J., Leicht, T., and Prill, F., "Discontinuous Galerkin methods for computational aerodynamics - 3D adaptive flow simulation with the DLR PADGE code," *Aerospace Science and Technology*, Vol. 14, No. 7, 2010, pp. 512–519.

[4]Galbraith, M. C., Benek, J. a., Orkwis, P. D., and Turner, M. G., "A Discontinuous Galerkin Chimera scheme," *Computers and Fluids*, Vol. 98, 2014, pp. 27–53.

[5]Orkwis, P. D. and McRae, D. S., "Newton's method solver for High-Speed Separated Flowfields," *AIAA Journal*, Vol. 30, No. 1, 1992, pp. 78–85.

[6]Galbraith, M. C., *A Discontinuous Galerkin Chimera Overset Solver*, Ph.D. thesis, University of Cincinnati, 2013.

[7]Galbraith, M., Allmaras, S., and Darmofal, D., "A Verification Driven Process for Rapid Development of CFD Software," *AIAA 2015-0818*, 2015.

[8]Griewank, A., *Evaluating derivatives : principles and techniques of algorithmic differentiation*, Society for Industrial and Applied Mathematics, 2000.

[9]Walther, A., Griewank, A., and Vogel, O., "ADOL-C: automatic differentiation using operator overloading in C++." *PAMM, Proc. Appl. Math. Mech.*, Vol. 2, No. 1, 2003, pp. 41–44.

[10]Bischof, C., Khademi, P., Mauer, A., and Carle, A., "Adifor 2.0: automatic differentiation of Fortran 77 programs," *IEEE computational science & engineering*, Vol. 3, No. 3, 1996, pp. 18–32.

[11]Pascual, V. and Hascoëet, L., "Extension of TAPENADE toward Fortran 95," *Automatic Differentiation: Applications, Theory, and Implementations*, Springer, 2006, pp. 171–179.

[12]Bischof, C. H., Hovland, P. D., and Norris, B., "On the implementation of automatic differentiation tools," *Higher-Order and Symbolic Computation*, Vol. 21, No. 3, 2008, pp. 311–331.

[13]Xia, Y., Luo, H., Frisbey, M., and Nourgaliev, R., "A set of parallel, implicit methods for a reconstructed discontinuous Galerkin method for compressible flows on 3D hybrid grids," *Computers and Fluids*, Vol. 98, 2014, pp. 134–151.

[14]Spall, R. E. and Yu, W., "Imbedded Dual-Number Automatic Differentiation for Computational Fluid Dynamics Sensitivity Analysis," *Journal of Fluids Engineering*, Vol. 135, No. 1, 2012, pp. 014501.

[15]Yu, W. and Blair, M., "DNAD, a simple tool for automatic differentiation of Fortran codes using dual numbers," *Computer Physics Communications*, Vol. 184, No. 5, 2013, pp. 1446–1452.

[16]Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., "The complex-step derivative approximation," *ACM Trans. Math. Softw.*, Vol. 29, No. 3, 2003, pp. 245–262.

[17]Kirby, R. M. and Karniadakis, G. E., "De-aliasing on non-uniform grids: algorithms and applications," *Journal of Computational Physics*, Vol. 191, No. 1, 2003, pp. 249–264.

[18]Wurst, M., Kessler, M., and Krämer, E., "IDIHOM: Industrialization of High-Order Methods - A Top-Down Approach," *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, Vol. 128, 2015, pp. 423–433.

[19]Roe, P. L., "The use of the Riemann problem in finite difference schemes," *Lecture Notes in Physics*, Vol. 141, No. 5, 1981, pp. 354–359.

[20]Bassi, F. and Rebay, S., "High-Order Accurate Discontinuous Finite Element Solution of the 2D Euler Equations," *Journal of Computational Physics*, Vol. 138, No. 2, 1997, pp. 251–285.

[21]Drela, M., "XFOIL: An Analysis and Design System for Low Reynolds Number Airfoils," *Low Reynolds Number Aerodynamics SE - 1*, edited by T. Mueller, Vol. 54 of *Lecture Notes in Engineering*, Springer Berlin Heidelberg, 1989, pp. 1–12.