Kedward, L. J., Allen, C. B., & Rendall, T. (2020). Comparing Matrix-based and Matrix-free Discrete Adjoint Approaches to the Euler Equations. In *AIAA SciTech Forum and Exposition, January 2020, Orlando, FL* [AIAA 2020-1294] American Institute of Aeronautics and Astronautics Inc. (AIAA). https://doi.org/10.2514/6.2020-1294

Peer reviewed version

Link to published version (if available):
10.2514/6.2020-1294

Link to publication record in Explore Bristol Research
PDF-document

This is the accepted author manuscript (AAM). The final published version (version of record) is available online via American Institute of Aeronautics and Astronautics, Inc. at 10.2514/6.2020-1294. Please refer to any applicable terms of use of the publisher.

University of Bristol - Explore Bristol Research

General rights

This document is made available in accordance with publisher policies. Please cite only the published version using the reference above. Full terms of use are available: http://www.bristol.ac.uk/red/research-policy/pure/user-guides/ebr-terms/

Comparing Matrix-based and Matrix-free Discrete Adjoint Approaches to the Euler Equations

L. J. Kedward* ; C. B. Allen† ; T. C. S. Rendall‡
Department of Aerospace Engineering, University of Bristol, Bristol, UK

Detail is presented on the implementation of numerical derivatives with focus given to the discrete adjoint equations. Two approaches are considered: a hybrid matrix-based scheme where the convective Jacobian is constructed explicitly; and a matrix-free method using reverse-mode automatic differentiation. The hybrid matrix-based scheme exploits a compact convective stencil using graph colouring to evaluate the convective Jacobian terms in $O(10)$ residual evaluations. Jacobian terms, grouped by colours, are evaluated using the complex step tangent model; this approach requires no external libraries or tools, minimal code modification and provides derivatives accurate to machine precision. The remaining artificial dissipation terms are trivial to differentiate by hand where the sensor coefficients are held constant. The hybrid matrix-based methodology is validated and compared with the 'traditional' matrix-free approach using reverse-mode automatic differentiation. The adjoint equations using both approaches are solved using the same fixed-point Runge-Kutta iteration accelerated by agglomeration multigrid. No loss in accuracy is seen between the matrix-based and the matrix-free methods when validated with the complex step tangent model. The hybrid matrix-based approach demonstrates a notable runtime performance advantage over the traditional matrix-free approach due to the prior calculation of Jacobian terms. Moreover, the convective Jacobian calculation takes less than 5% of primal runtime due to the compact stencil used. A critical analysis of the results and methodology is consequently presented, focussing on the general applicability of the hybrid approach to more complex problems.

I. Introduction

Derivatives are of widespread importance to numerical methods particularly those requiring Jacobian products and adjoint products. Jacobian products for example, are fundamental to the implementation of implicit Newton methods for the solution of difficult problems such as coupled physics or in the case of multiple length and time scales. The adjoint method has recently seen significant interest and development, most notably for error analysis[1], adaptive mesh refinement [2] and aerodynamic optimisation[3, 4].

Adjoint methods for fluid applications are typically categorised as either *continuous* or *discrete* corresponding to different ways of deriving the adjoint system (Figure 1). The continuous adjoint is derived from the linearised governing equations and then discretised for numerical solution, whereas the discrete adjoint is constructed from a linearisation of the discretised system. Early work within fluids favoured the continuous method due to the complexity associated with linearising numerical programs. In the early 1970s Pironneau applied the theory of optimal control for distributed parameter systems to obtain the continuous adjoint formulation for incompressible flows [5, 6]. In 1988 Jameson derived the continuous adjoint equations for inviscid compressible flow after which the application of the adjoint for aerodynamic shape optimisation became increasingly wide-spread. In 1998 Jameson, Martinelli and Pierce presented the first derivation of the continuous adjoint for the Navier-Stokes equations [3].

The solutions for the continuous and discrete adjoints should be equivalent in the limit of discretisation refinement, however there are usually minor differences in magnitude with much larger disagreement at sharp surface features. Importantly, only the discrete adjoint solution retains consistency with the primal flow solution, regardless of mesh

Copyright © 2019 by Laurence Kedward
*PhD Student, AIAA Student Member, laurence.kedward@bristol.ac.uk, Bristol, BS8 1TR, UK
†Professor of Computational Aerodynamics, AIAA Senior Member, c.b.allen@bristol.ac.uk, Bristol, BS8 1TR, UK
‡Senior Lecturer, AIAA Member, thomas.rendall@bristol.ac.uk, Bristol, BS8 1TR, UK

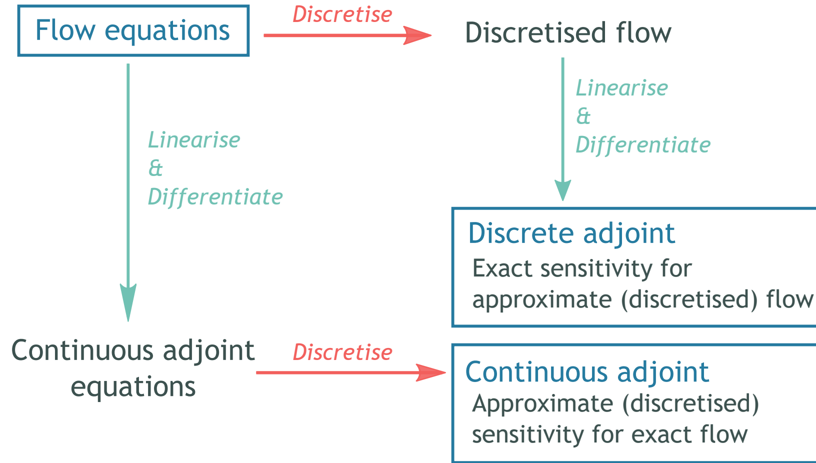


Fig. 1 Comparison between derivation of continuous and discrete adjoint formulations

resolution. This is important for shape optimisation in theoretically allowing exact convergence of optimality which is not possible by the continuous adjoint [7]. The inherent consistency of the discrete method means that adjoint boundary conditions are a result of the linearisation, whereas for the continuous case boundary conditions must also be derived for the adjoint problem. Moreover well-posed boundary conditions are not guaranteed to exist for all objective functions in the continuous approach [8]. Furthermore, the discrete adjoint typically inherits the convergence rate of the primal flow solution, however the same cannot be said for the continuous adjoint which may be ill-conditioned. An advantage of the continuous method is in its implementation; the continuous adjoint equations, derived from a linearisation of the flow equations, can be easily discretised and solved using the same numerical methods as for the flow. By contrast the discrete adjoint is implemented as a linearisation of the numerical program requiring either specialist automatic differentiation (AD) tools or a lengthy hand differentiation of the source code. Furthermore, the discrete adjoint has a high memory requirement, when compared to the primal or continuous solutions; as a result development of discrete adjoint methods has lagged that of the continuous with recent advances following corresponding developments in automatic differentiation technology and computational capability. Despite this, implementation of discrete adjoint codes remains an involved process, typically comprising a combination of automatic differentiation, hand differentiation and code tailoring.

In this paper, a review of numerical methods for derivatives is given with particular focus on the discrete adjoint method. This is followed by detail on the implementation of a hybrid method to the discrete adjoint which uses a matrix-based approach for the convective Jacobian and a matrix-free approach for dissipation terms. The sparse convective Jacobian matrix is constructed using tangent derivatives calculated using the complex step method and accelerated using graph colouring. The hybrid approach is implemented for the Euler equations discretised using a central scheme in an edge-based unstructured multigrid solver. Performance of the hybrid matrix-based approach is compared against the 'traditional' matrix-free method derived using reverse-mode automatic differentiation.

II. Background

In this section, background is given on the adjoint equations followed by detail on common approaches to program linearisation for discrete derivatives.

A. The discrete adjoint equations

The governing equations of the discretised flow are expressed:

$$\boldsymbol{R}(\boldsymbol{W}, \boldsymbol{X}) = 0 \tag{1}$$

where $\boldsymbol{R}$, $\boldsymbol{W}(\boldsymbol{\alpha})$ and $\boldsymbol{X}(\boldsymbol{\alpha})$ are the flow residual, conservative variables and computational mesh respectively, and $\boldsymbol{\alpha}$ is the vector of n_α design variables.

For some aerodynamic objective $J(\boldsymbol{W}, \boldsymbol{X})$, an augmented objective $I(\boldsymbol{W}, \boldsymbol{X})$ is formed using a set of Lagrange multipliers $\boldsymbol{\Lambda}$ to ensure satisfaction of the discrete flow equations:

$$I = J + \boldsymbol{\Lambda}^T \boldsymbol{R} \tag{2}$$

Now taking the total derivative of the augmented objective gives:

$$\frac{dI}{d\boldsymbol{\alpha}} = \frac{\partial J}{\partial \boldsymbol{X}}\frac{d\boldsymbol{X}}{d\boldsymbol{\alpha}} + \frac{\partial J}{\partial \boldsymbol{W}}\frac{d\boldsymbol{W}}{d\boldsymbol{\alpha}} + \boldsymbol{\Lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{W}}\frac{d\boldsymbol{W}}{d\boldsymbol{\alpha}} + \boldsymbol{\Lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{X}}\frac{d\boldsymbol{X}}{d\boldsymbol{\alpha}} \tag{3}$$

Rearranging this to factorise the term $d\boldsymbol{W}/d\boldsymbol{\alpha}$, which introduces dependency of the flow solution on the design variables, gives:

$$\frac{dI}{d\boldsymbol{\alpha}} = \underbrace{\left(\frac{\partial J}{\partial \boldsymbol{W}} + \boldsymbol{\Lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{W}}\right)}\frac{d\boldsymbol{W}}{d\boldsymbol{\alpha}} + \left(\frac{\partial J}{\partial \boldsymbol{X}} + \boldsymbol{\Lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{X}}\right)\frac{d\boldsymbol{X}}{d\boldsymbol{\alpha}} \tag{4}$$

From this it is clear that if the Lagrange multipliers are such that the bracketed term is zero, then the expensive term $d\boldsymbol{W}/d\boldsymbol{\alpha}$ can be avoided. Equating the bracketed term to zero therefore gives the discrete adjoint equation:

$$\frac{\partial J}{\partial \boldsymbol{W}} + \boldsymbol{\Lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{W}} = 0 \tag{5}$$

which is a linear system for the Lagrange multipliers, or adjoint state $\boldsymbol{\Lambda}$. Note that only the vector $\partial J/\partial \boldsymbol{W}$ is dependent on the specific objective function, not the flow Jacobian; implementation of different objectives simply requires substituting different partial derivatives for this right-hand side vector and within the total derivative calculation. It is the solution of this linear system that forms the core of adjoint solvers. Such solvers can be classified as either matrix-based or matrix-free; in the former case the Jacobian $\partial \boldsymbol{R}/\partial \boldsymbol{W}$ is constructed explicitly whereas in the latter case only adjoint-vector products are used. Additionally, a solver may use both matrix-free vector products and a matrix-based preconditioner, such as is required for Krylov solvers.

With a solution $\boldsymbol{\Lambda}$ to the adjoint equation, and a converged flow solution (*i.e.* $\boldsymbol{R} \approx 0$), the total derivative of the objective can be evaluated from the remaining non-zero terms of equation 4:

$$\frac{dJ}{d\boldsymbol{\alpha}} = \left(\frac{\partial J}{\partial \boldsymbol{X}} + \boldsymbol{\Lambda}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{X}}\right)\frac{d\boldsymbol{X}}{d\boldsymbol{\alpha}} \tag{6}$$

Implementing the discrete adjoint simply requires evaluation of the derivatives and adjoint products in equations 5 and 6; these are summarised in Table 1. All of these derivatives either involve a transpose Jacobian product or are derivatives of a single output with respect to many inputs; as is explained in the next section, this necessitates the need for the adjoint model (reverse mode) of program linearisation. Note that evaluation of the total derivative in equation 6 also involves the *mesh adjoint* product, which uses the mesh movement Jacobian ($\frac{d\boldsymbol{X}}{d\boldsymbol{\alpha}}$) to project the sensitivies from the grid vertices onto the design variables. This commonly involves two steps, first a projection onto the surface grid using the mesh movement adjoint, then a projection onto the design variables using the shape control method. For linear mesh movement and shape control schemes this projection is trivial [9], however non-linear methods require an additional linearisation of the mesh movement code [10].

In the following section, theory is presented on the two modes of program linearisation and the different ways of practically implementing them.

B. Program linearisation

A program or subroutine computes a function $\boldsymbol{F} : \mathbf{R}^n \to \mathbf{R}^m$ via a set of primitive logical operations on memory which, when abstracted, allow numerical calculation and program branching (*e.g.* `if`, `for`). Even in the presence of branching constructs a program with sufficient continuity permits a local linearisation:

$$\tilde{\boldsymbol{F}}(\boldsymbol{x}_0 + \boldsymbol{\Delta x}) = \boldsymbol{F}(\boldsymbol{x}_0) + A\boldsymbol{\Delta x} \tag{7}$$

where $A \in \boldsymbol{R}^{m\times n}$ is the Jacobian of $\boldsymbol{F}$ at $\boldsymbol{x}_0$. It is the linearisation provided by the Jacobian A and its products that are of significant importance across a variety of disciplines. In many such applications, the dimension of the Jacobian excludes the use of direct solution methods and instead iterative methods are usually employed. Excluding preconditioners, these iterative methods typically require only matrix-vector products of the Jacobian with which there are two types:

Table 1 Summary of derivatives for discrete adjoint

$\boldsymbol{\Lambda}^T(\partial \boldsymbol{R}/\partial \boldsymbol{W})$	Adjoint product	Jacobian of flow residual with respect to flow variables
$\boldsymbol{\Lambda}^T(\partial \boldsymbol{R}/\partial \boldsymbol{X})$	Adjoint product	Jacobian of flow residual with respect to grid vertices
$\frac{\partial J}{\partial \boldsymbol{W}}$	Partial derivative	Differentiation of objective with respect to flow variables
$\frac{\partial J}{\partial \boldsymbol{X}}$	Partial derivative	Differentiation of objective with respect to grid vertices
$\frac{d\boldsymbol{X}}{d\alpha}$	Total derivative	Differentiation of grid vertices with respect to design variables

1) **The tangent model:** also referred to as the *forward-mode* derivatives, this calculates a standard Jacobian product:

$$\dot{\boldsymbol{y}} = A\dot{\boldsymbol{x}} \tag{8}$$

2) **The adjoint model:** also referred to as the *reverse-mode* derivatives, this calculates the transpose vector product:

$$\bar{\boldsymbol{x}} = A^T\bar{\boldsymbol{y}} \tag{9}$$

In each case, the matrix-vector product operates on a *seed* vector $\dot{\boldsymbol{x}} \in \boldsymbol{R}^n$ or $\bar{\boldsymbol{y}} \in \boldsymbol{R}^m$ to provide a directional derivative. Here notation common to automatic differentiation literature is adopted where $\dot{\boldsymbol{y}}$ denotes derivatives of all outputs with respect to some combination of input perturbations and $\bar{\boldsymbol{x}}$ denotes some combination of output derivatives with respect to all inputs. These seed vectors are combinations of perturbed input and output states that are propagated through the program in the forward and reverse directions for the tangent and adjoint modes respectively.

When only Jacobian products are available, construction of the entire Jacobian can be performed by seeding with the cartesian basis vectors e_i which are entirely comprised of zeros except for a 1 in the ith component. For example, seeding with e_j gives the jth column of A when using the tangent Jacobian product and the jth row of A with the adjoint product. Clearly the number of matrix-vector products required to evaluate the entire Jacobian is n for the tangent mode and m for the adjoint mode. Put another way, the tangent mode provides the derivatives of all outputs with respect to a single input perturbation whereas the adjoint mode provides the derivatives of all inputs with respect to a single output perturbation. The tangent mode is therefore preferable when the number of outputs is much greater than the number of inputs and *vice versa* for the adjoint mode. For example, numerical optimisation problems are often formulated with a single objective function, between $O(1)$ and $O(10)$ non-linear constraints and between $O(10)$ and $O(100)$ design variables. In this case the number of inputs (design variables) almost always exceeds the number of outputs (objective function and constraints), and hence reverse mode derivatives are preferred. Alternatively, the implicit constraint for a converged flow state requires the solution of an adjoint system when calculating sensitivities, this is presented in the following section.

Several methods are available for evaluating tangent Jacobian products, each varying in accuracy, computational cost and ease of implementation. By contrast, evaluation of the adjoint product is considerably more complex, due to the need to reverse the call graph and save intermediate state, and is only possible with hand-derived code or automatic differentiation tools.

Finite differences

Perhaps the simplest and most intuitive method for tangent derivatives are finite differences. A first order forward-difference approximates the Jacobian product by the difference with a single perturbed state:

$$A_{\boldsymbol{x}_0}\dot{\boldsymbol{x}} \approx \frac{\boldsymbol{F}(\boldsymbol{x}_0 + \delta\dot{\boldsymbol{x}}) - \boldsymbol{F}(\boldsymbol{x}_0)}{\delta} \tag{10}$$

where δ is the step size. For second-order accuracy, a central-difference can be used, implementing the difference of two perturbed states:

$$A_{\boldsymbol{x}_0}\dot{\boldsymbol{x}} \approx \frac{\boldsymbol{F}(\boldsymbol{x}_0 + \delta\dot{\boldsymbol{x}}) - \boldsymbol{F}(\boldsymbol{x}_0 - \delta\dot{\boldsymbol{x}})}{2\delta} \tag{11}$$

In both cases, the choice of step size is of importance since it requires a trade-off between truncation error, which dominates as δ increases, and rounding error, which dominates as δ decreases [11]. Typically, if a function known to precision p, then first and second order step sizes can be chosen as $\sqrt{p}$ and $\sqrt[3]{p}$ respectively. Despite their simplicity

and limited accuracy, finite differences offer some powerful advantages over more complex methods, notably: they are *black-box* compatible, being entirely independent of the implementation of the target function and are hence trivial to evaluate in any situation; similarly, if multiple differences are required, they are perfectly parallel, and can hence make efficient use of HPC resources. A common application of finite differences is in matrix-free Newton Krylov methods whereby only matrix-vector products are required to solve the Newton system. The generality of finite differences allows the residual to be treated as a black-box, though degraded convergence may occur due to reduced accuracy.

Complex step

The lower bound restriction on finite difference step size occurs due to *catastrophic cancellation* in which the difference of two very similar numbers in finite precision floating point representation produces a loss of significant figures in the result. This can be overcome by the complex step method which requires modification of the program such that relevant inputs, outputs and intermediates are represented by a `complex` type. The tangent Jacobian product is then approximated by perturbing the imaginary component with the seed vector and taking the imaginary component of the resulting complex function value:

$$A_{\boldsymbol{x}_0}\dot{\boldsymbol{x}} \approx \frac{\mathrm{Im}\left[\boldsymbol{F}(\boldsymbol{x}_0 + i\delta\dot{\boldsymbol{x}})\right]}{\delta} \tag{12}$$

Not only is the complex step method second order accurate[12], but since there is no differencing involved the step size δ can be chosen much smaller than machine precision such that the result is effectively accuracy to machine precision. Slightly more expertise is required to implement the complex step method compared to finite differences, though it is still straightforward provided the source code is available and complex types are supported. Similarly, the runtime cost is higher for the complex code than for the original routine due to the complex numerics. This approach may be termed *grey-box* compatible since source code access is needed, but no external tool or significant re-write of the code is required. Custom functions may be required instead of intrinsics such as `min`, `max` and `abs` such that the complex variable code follows the same thread branch as the original.

The augmentation of the program with complex numbers bears resemblance to forward-mode automatic differentiation which introduces, for each relevant variable, a dual number comprising the variation thereof. The chain rule is then used to propagate the variations alongside the original code from the inputs to outputs. With the complex step method, these variations are stored in the imaginary component, which for sufficiently small step size has equivalent results as forward-mode AD. An important difference between forward mode AD and the complex step method however is that the complex step method performs redundant computations that eventually cancel to zero and is usually not as computationally efficient.

Automatic differentiation

Automatic differentiation, also known as algorithmic differentiation, describes methods by which the original program may be transformed or augmented to produce a second program capable of evaluating its derivatives, either in forward mode or reverse. As already mentioned, forward mode code can be generated by simply augmenting each variable and statement with a corresponding derivative propagation, the resulting derivatives are exact to machine precision. Whereas forward mode AD complements the forward methods already discussed, reverse-mode AD and equivalent hand-derived codes are the only options available for performing the adjoint product in a matrix-free manner. This highlights the complexity involved in evaluating the adjoint product. Reverse mode code, implementing the adjoint product, requires a forward pass to evaluate the original code and a reverse pass to back propagate derivatives. Moreover, certain data must be recorded or *taped* during the forward pass such that it is available during the reverse pass; this includes intermediate variables required in local partial derivatives, variables that are overwritten or incremented, and flags indicating the order of branching and looping. Various strategies are available for reducing the memory cost including check-pointing and recomputation, but application of reverse mode AD to large scale problems is restricted [13].

There are two main methods of implementing automatic differentiation tools:

1) **Operator-overloading:** relevant variables are replaced by a custom type which has its standard operations *overloaded* to allow automatic propagation of derivatives;
2) **Source code transformation:** a tool parses the primal code and produces new code which explicitly contains the propagation of derivatives.

For forward-mode differentiation, the difference between operator-overloading and source code transformation is only aesthetic since the overloaded operations can be inlined to reproduce the transformed code, however the same

cannot be said for the reverse mode. Operator-overloading constructs the reverse call graph as a stack of operations at runtime which is then interpreted in the reverse pass. By contrast, source code transformation tools explicitly produce reverse code which is then compiled normally and benefits from important optimisations performed by both the AD tool and compiler; hence adjoint codes produced by operator overloading are much slower and memory-intensive than those produced by source code transformation. Practical application of AD tools for the efficient generation of matrix-free adjoint products remains an involved process requiring expertise. Also worth mentioning is that the type and availability of automatic differentiation tools varies between programming languages. In particular, many specialist AD tools are not free and open source which severely restricts portability and security.

Sparse Jacobian

The methods discussed so far all provide matrix-free Jacobian products; that is, they do not require the evaluation and storage of the entire Jacobian matrix. This is advantageous in the general case since not only is there a cost associated with evaluating the entire Jacobian, but the storage cost can be unmanageable. An exception to this is if the Jacobian is sparse with some known structure, in which case storage of the Jacobian may be less than $O(10)$ times that of the state vector. In this work, a wrap-around approach to the discrete adjoint is presented using a sparse Jacobian constructed by the complex step method and accelerated by graph colouring as performed in [14] and [15]. In [14], Lyu *et al.* use forward-mode automatic differentiation with graph colouring to construct the sparse block Jacobian for the three-dimension Reynolds Averaged Navier Stokes equations. In [15] He *et al.* instead use finite differences to provide a more generic 'black-box' framework within openFOAM. The advantage of this method is that it avoids the complexity and development time associated with reverse mode automatic differentiation, while providing a very efficient adjoint product code.

III. Methodology

In this section, detail is presented on the unstructured Euler solver used here and the development of two discrete adjoint methods therein: a hybrid matrix-based approach, where the convective Jacobian is explicitly constructed, and a matrix-free approach using reverse-mode automatic differentiation for comparison and validation.

A. Flow discretisation and analysis

The compressible Euler equations are solved by a cell-centred finite volume method using the face-based methodology of Eliasson [16] for arbitrary shape elements. Compact central JST terms with artificial dissipation [17] are used for the discretisation with steady-state integration performed explicitly by four-stage Runge-Kutta. Convergence acceleration is performed using local time-stepping and non-linear agglomeration* multigrid [19].

B. Hybrid matrix-based discrete adjoint

For the hybrid matrix-based approach, the complex step method is used to construct a sparse Jacobian of the convective terms and the artificial dissipation terms are hand differentiated under a frozen coefficient assumption. Both the flow state Jacobian ($\partial\boldsymbol{R}/\partial\boldsymbol{W}$) and the flow grid metric Jacobian ($\partial\boldsymbol{R}/\partial\boldsymbol{X}$) are constructed in the same manner, however in this section descriptions are presented with respect to the flow state Jacobian only. The Jacobian matrix ($\partial\boldsymbol{R}/\partial\boldsymbol{W}$) has $n_{cell} \times n_w$ rows and columns where n_w is the number of conservative variables. To avoid evaluating the complex step residual $n_{cell} \times n_w$ times to construct each column separately, a graph colouring technique is used to exploit sparsity of the convective Jacobian.

Graph colouring

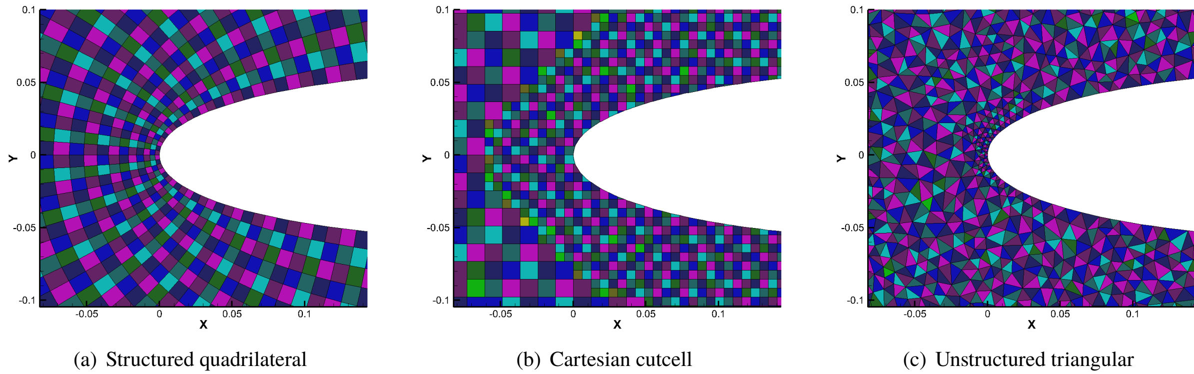
The compact stencils used here result in a highly sparse Jacobian for the convective terms with a known block structure. Consequently, graph colouring can be used to identify independent subsets of control volumes whereby control volumes of the same colour do not share cells in their stencils. These coloured sets correspond to structurally orthogonal columns in the Jacobian which do not have non-zeros in common rows. Therefore, if all control volumes of the same colour are perturbed simultaneously via the tangent model seed vector then the consequent output derivatives can be uniquely positioned within the non-zero structure of the Jacobian. With this, the number of complex step residual evaluations is only proportional to the number of colours required for the grid topology.

*Performed using MGridGen [18] library: http://glaros.dtc.umn.edu/gkhome/mgridgen/overview

A simple greedy colouring algorithm is adopted here, since the number of colours required for the convective stencil is only $O(10)$. This is outlined in algorithm 1; a loop over cells assigns colours by checking the cells in the local stencil and assigning the lowest integer which is not also assigned to any of the stencil cells. Figure 2 shows the resulting colour distribution for 3 different grid topologies.

Algorithm 1 Greedy graph colouring

1: **procedure**
2: *init:*
3:     **for** $cell$ **in** $cellList$ **do** ▷ Initialise cell colours to negative
4:         $cellColours(cell) \leftarrow -1$
5:     **end for**
6: *main:*
7:     **for** $cell$ **in** $cellList$ **do**
8:         $col \leftarrow 1$
9:         **while** $col \in cellColours(stencilCells(cell))$ **do** ▷ Find colour not equal to stencil neighbours
10:             $col \leftarrow col + 1$
11:         **end while**
12:         $cellColours(cell) \leftarrow col$ ▷ Set cell colour
13:     **end for**
14: **end procedure**



(a) Structured quadrilateral (b) Cartesian cutcell (c) Unstructured triangular

Fig. 2 Grid colouring

Jacobian construction and evaluation

Once the grid cells have been assigned colours, the Jacobian can be constructed by performing complex step evaluations of the flow convective residual for each colour. Algorithm 2 outlines the process required to construct the sparse Jacobian using the complex-step method and graph colouring.

The number of complex residual evaluations required is $n_w \times n_{colour}$. Despite the higher cost of the complex residual routine compared to equivalent forward mode AD code, this is only required during Jacobian construction which occurs prior to the much more computationally demanding solve of the adjoint system. Line 13 in the algorithm denotes saving the resulting derivative vector $\boldsymbol{dR}$ into a sparse structure; this requires identifying which components of the derivative vector belong to which columns and rows based on the current colour. Once identified, this operation places the components directly into a sparse structure such that storage for the full Jacobian is never allocated. The sparse structure requires memory proportional to the number of non-zero values which is:

$$n_{nnz} = (n_w)^2 \cdot \sum_{i=1}^{n_{cell}} m_i \tag{13}$$

Algorithm 2 Construction of sparse Jacobian using complex step

```
 1: procedure
 2:     for colour in colourList do                         ▷ Loop over colours
 3:         for i = 1, n_w do                               ▷ Loop over conservative variables
 4:             for cell in cellList do
 5:                 if cellColours(cell) == colour then
 6:                     ẇ(i, cell) ← 0 + iδ                 ▷ Construct complex step seed
 7:                 else
 8:                     ẇ(i, cell) ← 0 + 0i
 9:                 end if
10:             end for
11:             R ← Residual(W + ẇ)                         ▷ Evaluate residual
12:             dR ← Imag(R)/δ                              ▷ Perform complex step
13:             sparseJacobian ← dR                         ▷ Save to sparse data structure
14:         end for
15:     end for
16: end procedure
```

where m_i is the number of cells within the convective stencil of cell i. Assuming a grid of uniform topology where all grid cells have at most $\bar{m}$ stencil cells, then the number of non-zero values is bounded by:

$$n_{nnz} \leq n_w^2 \cdot \bar{m} \cdot n_{cell} = O(n_{cell}) \tag{14}$$

i.e. the memory cost of storing the sparse Jacobian is linear with grid size. Despite this, the memory cost is sensitive to the parameter $\bar{m}$ which represents the size of the influence stencil. In this work, only the convective Jacobian is constructed which has a compact stencil due to the central scheme used. Dissipation contributions have been hand-differentiated and are hence evaluated in matrix-free fashion. This represents a *hybrid* approach to adjoint evaluation involving both matrix-based and matrix-free operators.

C. Matrix-free discrete adjoint

For validation and comparison, a 'traditional' matrix-free adjoint code is produced using automatic differentiation[†]. Once again, only the convective terms need differentiation with AD since the dissipation terms have already been differentiated by hand. The AD tool was used with minimal code modification using the default settings which include checkpointing, recomputation and optimisation of the derivative code. Listing 1 gives an extract from the face flux calculation. Listing 2 gives the corresponding reverse-mode adjoint code produced by automatic differentiation; the addition of 'b' to variable names indicates the corresponding derivative variable.

[†]Performed using Tapenade[20]: https://www-sop.inria.fr/tropics/tapenade.html

Listing 1 Flux calculation extract

```
nvel = u*nx + v*ny + w*nz
flx(1) = rho*nvel
flx(2) = rho*u*nvel + p*nx
flx(3) = rho*v*nvel + p*ny
flx(4) = rho*w*nvel + p*nz
flx(5) = rho*nvel*e + nvel*p
```

Listing 2 Flux derivative code extract

```
rhob = e*nvel*flxb(5)
nvelb = (p+e*rho)*flxb(5)
eb = rho*nvel*flxb(5)
pb = nvel*flxb(5)
rhob = rhob + nvel*w*flxb(4)
wb = nvel*rho*flxb(4)
nvelb = nvelb + rho*w*flxb(4)
pb = pb + nz*flxb(4)
rhob = rhob + nvel*v*flxb(3)
vb = nvel*rho*flxb(3)
nvelb = nvelb + rho*v*flxb(3)
pb = pb + ny*flxb(3)
rhob = rhob + nvel*u*flxb(2)
ub = nvel*rho*flxb(2)
nvelb = nvelb + rho*u*flxb(2)
pb = pb + nx*flxb(2)
rhob = rhob + nvel*flxb(1)
nvelb = nvelb + rho*flxb(1)
ub = ub + nx*nvelb
vb = vb + ny*nvelb
wb = wb + nz*nvelb
```

D. Explicit multigrid solver

Solution of the discrete adjoint system is performed using the same non-linear iteration as for the primal flow solution: in this work an explicit multistage Runge-Kutta scheme accelerated by agglomeration multigrid. This allows code-reuse and minimises development time while also guaranteeing a minimum convergence rate bounded by the primal problem [8]. Applied to a linear system, the multistage Runge-Kutta scheme applies simple corrections using a residual of the form:

$$\boldsymbol{R_{adj}} = A^T \boldsymbol{\Lambda} + D^T \boldsymbol{\Lambda} - \boldsymbol{b} \tag{15}$$

where $A^T \boldsymbol{\Lambda}$ is the the convective Jacobian adjoint product, $D^T \boldsymbol{\Lambda}$ is the hand-differentiated matrix-free adjoint dissipation operator and $\boldsymbol{b}$ is the right-hand side of the adjoint equation $-\partial J/\partial \boldsymbol{W}$.

The convective Jacobian adjoint product ($A^T \boldsymbol{\Lambda}$) is evaluated using either an efficient sparse-vector multiplication in the case of the matrix-based method or by calling the derivative code produced by automatic differentiation for the matrix-free method.

As with the primal flow, agglomeration multigrid is used to accelerate convergence of the adjoint solution. Unlike the primal flow, which uses a non-linear full approximation storage scheme, the adjoint multigrid uses a simple linear correction scheme. For the matrix-based method, the convective adjoint operator on the coarse grids involves constructing coarse grid Jacobians using the Galerkin method based on the prolongation and restriction operators I_p and I_r:

$$A_{coarse} = I_r \cdot A_{fine} \cdot I_p \tag{16}$$

These sparse products are easily performed and result in a very efficient coarse grid operator. By contrast, for the dissipation terms and the matrix-free method, the coarse grid operator is constructed using the re-discretisation method, whereby the derivative code is called on the coarse grid topology. The former methodology (Galerkin) represents the coarse grid approximation to the adjoint problem, whereas the latter is the adjoint of the coarse grid discretisation [19]. Note that on the coarse grids a simpler first order dissipation operator is used in place of the usual second and fourth order dissipation, as outlined by Eliasson [16]; this applies to both the primal and adjoint solutions.

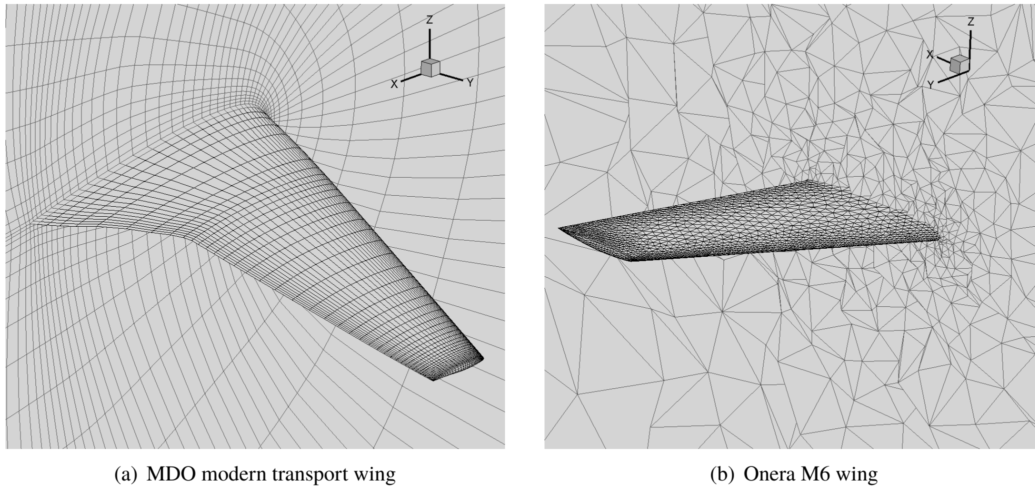
IV. Results

A. Test grids

Two coarse test grids are used for validation and performance testing: a structured hexahedral mesh around the MDO wing[21], generated by transfinite interpolation [22] with improved orthogonality and smoothness; and an unstructured tetrahedral mesh for the Onera M6 wing[23]. Grid sizes and run conditions are given in Table 2. Coarse grids are used in this study to ensure tight convergence and for a reasonable runtime since all tests are run in serial.

Table 2 Test cases used for validation and performance evaluation

	MDO Transport Wing (Coarse)	Onera M6 Wing (Coarse)
Cell type	Hexahedral	Tetrahedral
Surface grid points	3881	4794
Volume grid cells	125K	300K
Mach number	0.85	0.84
Target CL	0.47	0.26



(a) MDO modern transport wing

(b) Onera M6 wing

Fig. 3 Surface and volume views for test case meshes

B. Derivative validation

Validation of the derivative codes can be done at two levels: at the adjoint-vector product level and at the objective function level.

Jacobian inner products

Returning to equations 8 and 9 for the definition of the tangent and adjoint models respectively, the following identity can be written:

$$\bar{\mathbf{x}}^T \mathbf{b} = \mathbf{b}^T \dot{\boldsymbol{y}} \tag{17}$$

for any seed vector $\mathbf{b}$, where $\bar{\boldsymbol{x}}$ is the result of the adjoint product with $\boldsymbol{b}$ and $\dot{\boldsymbol{y}}$ results from the tangent product with $\boldsymbol{b}$. This is useful in that it allows the adjoint model to be validated using the tangent model. Shown in Table 3 are numerical results using a random unit vector for $\boldsymbol{b}$. The adjoint product is evaluated using both the matrix-based method,

Table 3 Inner product validation of adjoint models

$\bar{\mathbf{x}}^T\mathbf{b}$	Complex-step Jacobian	0.60555704713193**3**
	AD Adjoint	0.60555704713193**6**
$\mathbf{b}^T\dot{y}$	Complex step	0.605557047131934
	Finite difference	0.60555**6501974470**

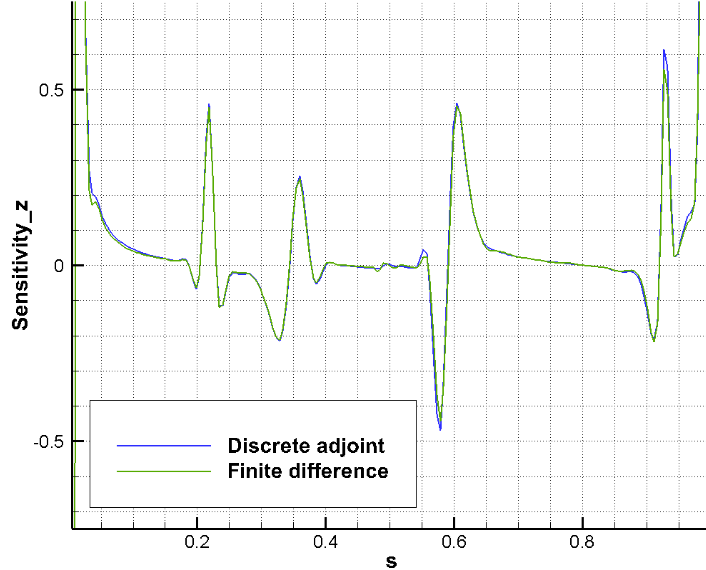


Fig. 4 Comparison between finite difference and discrete adjoint for drag sensitivities

constructed using the complex step, and the matrix-free method, constructed using automatic differentiation. The tangent model is evaluated using the complex step method and finite differences. The matrix-based and the matrix free adjoint products agree with the complex step tangent model to within 1×10^{-14}. As is to be expected, the tangent model using finite differences is only accurate to 1×10^{-6} corresponding to the rounding error in double precision arithmetic.

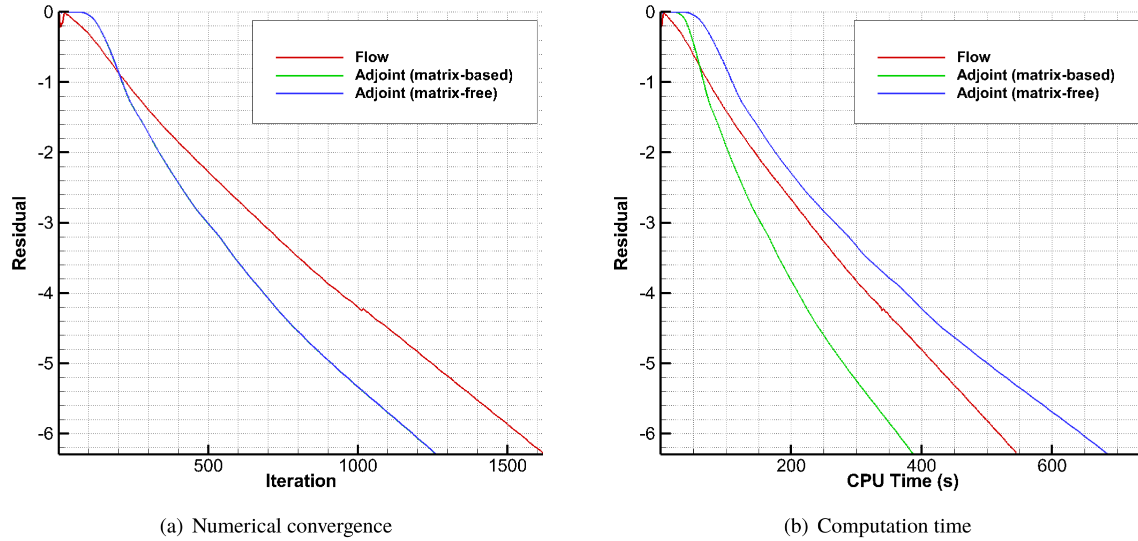
Objective function gradient

A higher level validation can be made using the total derivative of the objective given by equation 6. The total derivative at each point produced from the adjoint solution can be compared to finite differences. Since this requires as many flow evaluations as surface points, this validation is run on a 2D aerofoil case. Figure 4 shows the z-ordinate drag sensitivity calculated both by finite difference and from the discrete adjoint implemented here. Good agreement is seen in the general trend across the surface where the discrepancies between the two are dominated by the truncation error during finite differencing.

C. Performance

Figures 5(a) and 6(a) show convergence histories for both the MDO and M6 wing test cases. Both cases are converged to a residual decrease of six orders of magnitude for both the primal and adjoint problems. In both cases, the asymptotic convergence rate of the adjoint problem matches that of the primal. Note that both adjoint strategies (matrix-based and matrix-free) converge identically. Recall that whereas both fine grid adjoint operators have been shown to be equivalent (Table 3), their respective coarse grid operators were implemented differently. It is clear however, that the choice of coarse grid operator (Galerkin or re-discretisation) does not have any major effect on the overall convergence rate for the Euler equations.

Despite having identical convergence rates, the realtime performance, shown in Figures 5(b) and 6(b), differs



(a) Numerical convergence

(b) Computation time

Fig. 5 Convergence of the primal and adjoint solutions for the MDO test case

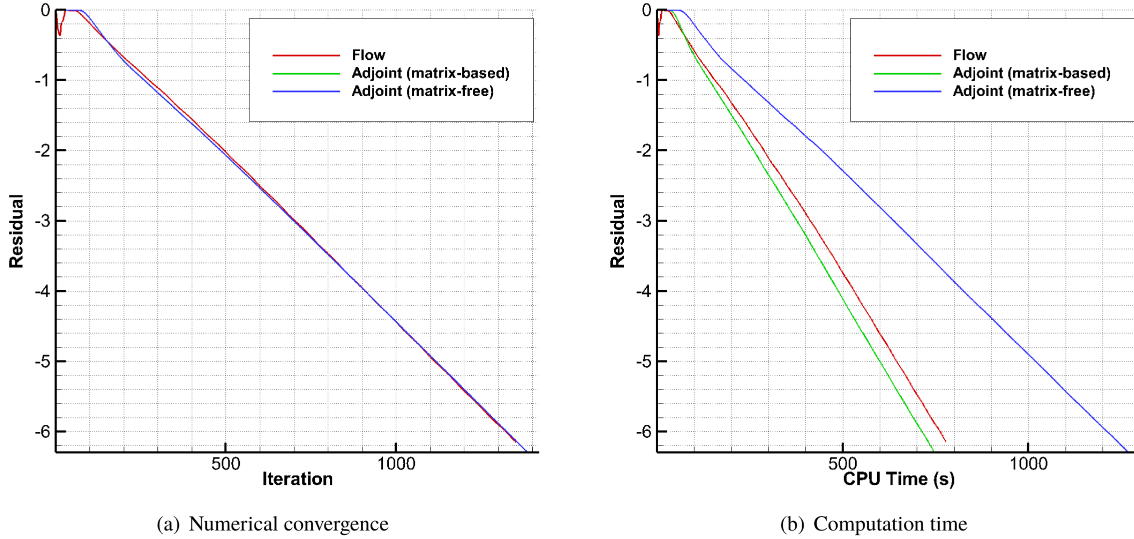
between the two adjoint implementations. The matrix-based method actually converges faster than the primal solution in both cases due to the efficient sparse-matrix products. For the structured hexahedral MDO case, the matrix-based adjoint is approximately three-quarters the time of the flow solution. Moreover, construction of the corresponding convective Jacobian is fast, requiring less than 5% the time of a flow solution in both cases. The structured hexahedral mesh requires 13 colours for the grid colouring, with the tetrahedral mesh requiring 11 colours due to the fewer stencil cells. Therefore the entire convective Jacobian is constructed in only 65 and 55 complex residual evaluations for the MDO case and M6 case respectively.

By comparison the matrix-free adjoint solution using automatic differentiation requires between 30% and 70% more time than a single flow solution. While being slower than the matrix-based method shown here, this is still an impressive performance for a reverse-mode code and demonstrates the efficiency of automatic differentiation using source-code transformation. By comparison, the test cases considered in [20] show adjoint runtime ratios between 2 and 16 when using the same AD tool as here. Runtime ratios and breakdown are given in Table 4.

Table 4 Single-core runtime, normalised by the respective flow runtime for each case

	MDO	M6
Flow solution	1.00	1.00
Matrix-based adjoint		
Jacobian construction	0.02	0.04
Adjoint solution	0.77	0.98
Matrix-free adjoint		
Adjoint solution	1.36	1.67

Table 5 gives the memory usage and the ratio with respect to that of the primal solver for each case. As is expected the matrix-based method has a larger memory requirement than the matrix-free method in order store the sparse convective Jacobian. The hexahedral test mesh requires almost four times as much memory as the primal solver, compared to the tetrahedral mesh which requires 3.75 times that of the primal; this can be attributed to the denser Jacobian in the hexahedral case.



(a) Numerical convergence

(b) Computation time

Fig. 6 Convergence of the primal and adjoint solutions for the Onera M6 test case

Table 5 Total memory usage

	MDO		M6	
	MBytes	Ratio	MBytes	Ratio
Flow	133	1.00	237	1.00
Matrix-based adjoint	528	3.97	888	3.75
Matrix-free adjoint	219	1.65	382	1.61

D. Discussion

Compared to previous matrix-based adjoint implementations[14, 15], the approach presented in this paper is a hybrid methodology where only the compact terms of the Jacobian have been stored; the dissipation terms, with a much larger stencil, were differentiated by hand and evaluated in a matrix-free manner when solving the adjoint equations. The resulting adjoint product is very computationally efficient when compared to a purely matrix-free adjoint product at the cost of a moderate increase in memory usage. These results take advantage of the JST scheme[17] which has a very compact convective stencil, and a simple artificial dissipation scheme. Moreover, the use of a four-stage Runge-Kutta scheme for the fixed-point iteration means that the costly dissipation fluxes, and the respective adjoints, are only evaluated once for every four convective flux/adjoint evaluations.

Clearly a disadvantage of this hybrid methodology is how tailored it is to specific discretisation schemes and numerical methods. Application of the same scheme to an upwind method using MUSCL for example would result in a larger convective stencil and therefore an increase in memory required for the Jacobian. By comparison the matrix-free method using automatic differentiation is very flexible with respect to the numerical problem at hand; runtime performance, however, is very sensitive to the code implementation.

Not investigated in this work is the choice of linear solver. Using the same fixed-point iteration as the primal with linear multigrid acceleration was sufficient and straightforward to implemented here, however the use of Krylov methods such as GMRES is increasingly common within aerospace for implicit solvers and their adjoints. In this respect the matrix-based methodology may offer some advantages in simplifying the construction of the preconditioners required for Krylov methods.

V. Conclusion

In this paper, detail has been presented on the evaluation of derivatives for the discrete adjoint equations and a hybrid matrix-based methodology has been presented for the Euler adjoint equations. The hybrid matrix-based scheme exploits a compact convective stencil and uses graph colouring to evaluate the convective Jacobian terms in $O(10)$ residual evaluations. The complex step is used to evaluate Jacobian columns, grouped by colours, in tangent mode. This approach requires no external libraries or tools, minimal code modification and provides derivatives accurate to machine precision. The remaining artificial dissipation terms are implemented in a matrix-free manner using hand-differentiated code under the assumption of frozen sensor coefficients.

The hybrid matrix-based methodology has been validated and compared with the 'traditional' matrix-free approach using reverse-mode automatic differentiation. Both the hybrid matrix-based and traditional matrix-free methods have been validated using the complex step tangent model and no loss in accuracy is seen in either method. The hybrid matrix-based approach has demonstrated a notable runtime performance advantage over the traditional matrix-free approach due to the ahead-of-time calculation of the convective Jacobian terms such that only sparse matrix-vector products are required when solving the linear system. Moreover, calculation of the convective Jacobian terms takes less than 5% of primal runtime due to the compact stencil used.

Critical analysis of the hybrid matrix-based methodology highlights a disadvantage in comparison to the traditional matrix-free approach in that the current methodology requires a degree of tailoring to the specific numerical scheme. By comparison the use of mature automatic differentiation tools means that the matrix-free methodology can be implemented with minimal consideration of the underlying numerics. That said, the use of automatic differentiation tools remains an involved task, still requiring some manual optimisation and tailoring of codes. Ongoing work will consider automated and more general approaches to ahead-of-time evaluation of compact Jacobian terms used in combination with matrix-free methods for non-compact terms.

References

[1] Nemec, M., and Aftosmis, M., "Adjoint Error Estimation and Adaptive Refinement for Embedded-Boundary Cartesian Meshes," *18th AIAA Computational Fluid Dynamics Conference*, 2007. doi:10.2514/6.2007-4187.

[2] Nemec, M., Aftosmis, M., and Wintzer, M., "Adjoint-Based Adaptive Mesh Refinement for Complex Geometries," *46th AIAA Aerospace Sciences Meeting and Exhibit*, 2008. doi:10.2514/6.2008-725.

[3] Jameson, A., Martinelli, L., and Pierce, N., "Optimum Aerodynamic Design Using the Navier-Stokes Equations," *Theoretical and Computational Fluid Dynamics*, Vol. 10, No. 1-4, 1998, pp. 213–237. doi:10.1007/s001620050060.

[4] Mani, K., and Mavriplis, D. J., "Adjoint-Based Sensitivity Formulation for Fully Coupled Unsteady Aeroelasticity Problems," *AIAA Journal*, Vol. 47, No. 8, 2009, pp. 1902–1915. doi:10.2514/1.40582.

[5] Pironneau, O., "On optimum profiles in Stokes flow," *Journal of Fluid Mechanics*, Vol. 59, No. 1, 1973, pp. 117–128.

[6] Pironneau, O., "On optimum design in fluid mechanics," *Journal of Fluid Mec*, Vol. 641, No. 1, 1974, pp. 979–110.

[7] Giles, M. B., and Pierce, N. A., "An Introduction to the Adjoint Approach to Design," *Flow, Turbulence and Combustion*, Vol. 65, 2000, pp. 393–415.

[8] Peter, J. E., and Dwight, R. P., "Numerical sensitivity analysis for aerodynamic optimization: A survey of approaches," *Computers and Fluids*, Vol. 39, No. 3, 2010, pp. 373–391. doi:10.1016/j.compfluid.2009.09.013.

[9] Poirier, V., and Nadarajah, S., "Efficient RBF Mesh Deformation Within An Adjoint-Based Aerodynamic Optimization Framework," *Journal of Aircraft*, Vol. 53, No. 6, 2016, pp. 1–19. doi:10.2514/1.C033573.

[10] Nielsen, E. J., and Park, M. a., "Using an Adjoint Approach to Eliminate Mesh Sensitivities in Computational Design," *AIAA Journal*, Vol. 44, No. 5, 2006, pp. 948–953. doi:10.2514/1.16052.

[11] Haftka, R., "Selecting step sizes in sensitivity analysis by finite differences," Tech. Rep. Technical Memorandum 86382, NASA, 1985.

[12] Martins, J. R. R. A., Studza, P., and Alonso, J. J., "The Complex-Step Derivative Approximation," *ACM Transactions on Mathematical Software*, Vol. 29, No. 3, 2003, pp. 245–262.

[13] Mader, C. A., R. A. Martins, J. R., Alonso, J. J., and Der Weide, E. V., "ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers," *AIAA Journal*, Vol. 46, No. 4, 2008, pp. 863–873. doi:10.2514/1.29123.

[14] Lyu, Z., Kenway, G. K., Paige, C., and Martins, J., "Automatic Differentiation Adjoint of the Reynolds-Averaged Navier-Stokes Equations with a Turbulence Model," *21st AIAA Computational Fluid Dynamics Conference*, San Diego, California, 2013. doi:10.2514/6.2013-2581.

[15] He, P., Mader, C. A., Martins, J., and Maki, K., "An Object-oriented Framework for Rapid Discrete Adjoint Development using OpenFOAM," *AIAA Scitech 2019 Forum*, San Diego, California, 2019. doi:10.2514/6.2019-1210.

[16] Eliasson, P., "EDGE, a Navier-Stokes Solver for Unstructured Grids," Tech. rep., FOI, 2002.

[17] Jameson, A., Schmidt, W., and Turkel, E., "Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes," *14th Fluid and Plasma Dynamics Conference*, 1981.

[18] Moulitsas, I., and Karypis, G., "Multilevel Algorithms for Generating Coarse Grids for Multigrid Methods," *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, ACM, New York, NY, USA, 2001, pp. 45–45. doi:10.1145/582034.582079.

[19] Mavriplis, D. J., "Multigrid Techniques for unstructured Meshes," *26th Computational Fluid Dynamics Lecture Series Program of the von Karman Institute (VKI) for Fluid Dynamics*, 1995.

[20] Hascoët, L., and Pascual, V., "The Tapenade Automatic Differentiation tool: Principles, Model, and Specification," *ACM Transactions On Mathematical Software*, Vol. 39, No. 3, 2013. doi:10.1145/2450153.2450158.

[21] Allwright, S., "Multi-discipline optimisation in preliminary design of commercial transport aircraft," *Computational Methods in Applied Sciences '96, ECCOMAS*, 1996.

[22] Allen, C. B., "Towards automatic structured multiblock mesh generation using improved transfinite interpolation," *International Journal for Numerical Methods in Engineering*, Vol. 74, No. 5, 2008, pp. 697–733. doi:10.1002/nme.2170.

[23] Schmitt, V., and Charpin, F., "Pressure distributions on the ONERA M6 wing at transonic Mach numbers," Tech. rep., Fluid Dynamics Panel Working Group 04, AGARD AR 138, May 1979.