Efficient Numerical Differentiation of Implicitly-Defined Curves for Sparse Systems

David A. Brown and David W. Zingg

University of Toronto Institute for Aerospace Studies, Toronto, Ontario, M3H 5T6, Canada

Abstract

A numerical technique is developed for the efficient numerical differentiation of regular implicitly-defined curves existing in high-dimensional real space such as those representing homotopies, where the system of equations which defines the curve implicitly is assumed to be sparse. The calculation is verified numerically through its application to the curve defined implicitly by a homotopy constructed based on a discretization of the equations governing compressible aerodynamic fluid flow. Consideration is given to computational cost, data storage, and accuracy. This method is applicable to any implicitly-defined curves or trajectories which can occur, for example, in dynamical systems analysis or control. Applications also exist in the area of homotopy continuation where implicitly-defined curves are approximately traced numerically. Such applications include the analysis of curve traceability and the construction of higher order predictors. The latter is investigated numerically and it is found that increasing the order of accuracy of the predictor can significantly improve the curve-tracing accuracy within a limited radius.

Keywords: numerical differentiation, implicitly-defined curves, homotopy, continuation, high-order predictor, sparse systems

1. Introduction

Consider a curve segment defined implicitly by the system of equations

$$\mathcal{H}\left(\mathbf{q}\left(\lambda\right),\lambda\right)=\mathbf{0}, \tag{1}$$

$\mathcal{H}:\mathbb{R}^N\times\mathbb{R}\rightarrow\mathbb{R}^N$, $\mathbf{q}\in\mathbb{R}^N$, $\lambda\in\mathbb{R}$ on some interval $\lambda\in\Lambda$, $\Lambda\subset\mathbb{R}$. Without loss of generality, let $\Lambda=\{[0,1]\}$, and assume that the curve is oriented in the direction of decreasing λ. In this paper we assume that $\mathcal{H}$ is at least C^1 differentiable, invertible, and that the curve is regular on Λ. As such, the curve derivatives cannot vanish and no bifurcations are present. Hence, equation (1) is said to describe a regular homotopy [1] in $\mathbb{R}^N$. While homotopies including bifurcations have garnered much interest for the study of systems of equations of multiple solutions [19, 31, 33, 38, 39, 40, 41], and we do recognize the importance of including consideration for such points, they fall outside the scope of our particular applications and hence the scope of this paper.

It may be of practical interest to calculate higher derivatives of the curve, either for analysis or for application to numerical algorithms. While such calculations have been performed previously [29, 34, 36], the authors have not included special consideration for sparsity and the

calculations can become prohibitively expensive if $\mathcal{H}$ is large and sparse. It is also important that the calculations be efficient if the calculation is to be used as part of a cost-competitive continuation algorithm. An example where this is important is homotopy continuation [1], which we have been developing as an efficient continuation strategy for solving the sparse algebraic systems of equations arising in computational fluid dynamics (CFD) problems [5, 3, 4]. These systems of equations are sparse and it is not uncommon for the equations to number in the tens of millions, or even higher in some applications.

The calculations in this paper are developed in a Jacobian-free Krylov framework using the flexible generalized minimal residual (FGMRES) [32] method, though any linear solver suitable for solving linear systems of the form

$$\nabla_{\mathbf{q}}\mathcal{H}\mathbf{x} = \mathbf{b} \tag{2}$$

could be used, where $\mathbf{x}, \mathbf{b} \in \mathbb{R}^N$ and $\nabla\mathcal{H}$ indicates the Jacobian[1] of $\mathcal{H}$. The distinction that we are making with equation (2) is that the linear system is represented by a Jacobian matrix. Some linear solvers can make use of approximate Jacobian-vector products to avoid forming the Jacobian matrix explicitly.

2. Tangent Vector

Consider the curve defined implicitly by a regular homotopy

$$\mathcal{H}\left(c\left(s\right)\right) = \mathbf{0}, \tag{3}$$

where the curve $c\left(s\right) = \left(\mathbf{q}\left(s\right); \lambda\left(s\right)\right)$, $c : \mathbb{R} \to \mathbb{R}^N \times \mathbb{R}$ has an arclength parametrization [17] defined implicitly by

$$\dot{c}\left(s\right) \cdot \dot{c}\left(s\right) = 1, \tag{4}$$

$s \in \mathcal{S}$, $\mathcal{S} \subset \mathbb{R}$, $\mathcal{S} = \{[0, s_{\text{tot}}]\}$. Differentiating both sides of equation (3) with respect to the arclength parameter s gives:

$$\nabla\mathcal{H}\left(c\left(s\right)\right)\dot{c}\left(s\right) = \mathbf{0}, \tag{5}$$

which can also be written:

$$\nabla_{\mathbf{q}}\mathcal{H}\left(c\left(s\right)\right)\dot{\mathbf{q}}\left(s\right) + \dot{\lambda}\left(s\right)\frac{\partial}{\partial\lambda}\mathcal{H}\left(c\left(s\right)\right) = \mathbf{0}, \tag{6}$$

or, after rearranging:

$$\nabla_{\mathbf{q}}\mathcal{H}\left(c\left(s\right)\right)\left[\frac{-1}{\dot{\lambda}\left(s\right)}\dot{\mathbf{q}}\left(s\right)\right] = \frac{\partial}{\partial\lambda}\mathcal{H}\left(c\left(s\right)\right). \tag{7}$$

Define the vector $\mathbf{z} \in \mathbb{R}^N$ such that

$$\nabla_{\mathbf{q}}\mathcal{H}\left(c\left(s\right)\right)\mathbf{z} = \frac{\partial}{\partial\lambda}\mathcal{H}\left(c\left(s\right)\right). \tag{8}$$

[1] When ∇ appears without subscript, differentiation is performed with respect to all variables including λ. The notation $\nabla_{\mathbf{q}}$ means that differentiation is with respect to the vector $\mathbf{q}$ only.

Then

$$\dot{\mathbf{q}} = -\dot{\lambda}\mathbf{z} \tag{9}$$

and

$$\dot{c}(s) \cdot \dot{c}(s) = \dot{\mathbf{q}}(s) \cdot \dot{\mathbf{q}}(s) + \dot{\lambda}(s)\,\dot{\lambda}(s) = \dot{\lambda}^2 [\mathbf{z} \cdot \mathbf{z} + 1]. \tag{10}$$

Since $\dot{c}(s) \cdot \dot{c}(s) = 1$, equation (10) can be used to obtain an equation for $\dot{\lambda}(s)$:

$$\dot{\lambda}(s) = \frac{-1}{\sqrt{\mathbf{z} \cdot \mathbf{z} + 1}}, \tag{11}$$

where the negative sign has been included to force a negative orientation for $\dot{\lambda}(s)$, which is the convention that we have adopted. Substituting this back into equation (7) gives the expression for $\dot{\mathbf{q}}(s)$:

$$\dot{\mathbf{q}}(s) = -\dot{\lambda}\mathbf{z}. \tag{12}$$

The tangent vector can thus be calculated from equations (11) and (12), where $\mathbf{z}$ is defined by equation (8) and requires the solution to a sparse linear system of equations.

3. Curvature Vector

As with the tangent vector, the curvature vector will depend on the parametrization. Carrying over from the tangent calculation, an arclength parametrization is assumed. The derivation begins by differentiating both sides of equation (5), which gives

$$\nabla\mathcal{H}(c(s))\,\ddot{c}(s) + \nabla^2\mathcal{H}(c(s))[\dot{c}(s), \dot{c}(s)] = \mathbf{0}. \tag{13}$$

For clarity of presentation, let $\mathbf{w}_2 = \nabla^2\mathcal{H}(c(s))[\dot{c}(s), \dot{c}(s)]$, $\mathbf{w}_2 \in \mathbb{R}^N$. Methods for approximating tensor-vector products are discussed in Section 5. Equation (13) can be expanded to

$$\nabla_{\mathbf{q}}\mathcal{H}(c(s))\,\ddot{\mathbf{q}}(s) + \ddot{\lambda}\frac{\partial}{\partial\lambda}\mathcal{H}(c(s)) = -\mathbf{w}_2. \tag{14}$$

Equation (8) can be used to simplify:

$$\nabla_{\mathbf{q}}\mathcal{H}(c(s))\left[\ddot{\mathbf{q}} + \ddot{\lambda}\mathbf{z}\right] = -\mathbf{w}_2. \tag{15}$$

Let

$$\mathbf{z}_2 = \ddot{\mathbf{q}} + \ddot{\lambda}\mathbf{z}, \tag{16}$$

$\mathbf{z}_2 \in \mathbb{R}^N$. It is possible to solve the linear system (15) for $\mathbf{z}_2$. However, an additional equation is needed to retrieve all $N + 1$ initial unknowns. As with the tangent calculation, this equation comes from the parametrization. Differentiating both sides of the arclength definition given by equation (4) gives the new equation

$$\ddot{c}(s) \cdot \dot{c}(s) = 0 \tag{17}$$

which, when expanded, can be written in terms of $\ddot{\mathbf{q}}$, $\ddot{\lambda}$, and the vector $\mathbf{z}$ previously calculated during the tangent calculation:

$$\ddot{\mathbf{q}} \cdot \mathbf{z} - \ddot{\lambda} = 0. \tag{18}$$

To solve for $\ddot{\lambda}$, take the dot product $\mathbf{z}_2 \cdot \mathbf{z}$ and use equations (16) and (18):

$$\mathbf{z}_2 \cdot \mathbf{z} = \ddot{\mathbf{q}} \cdot \mathbf{z} + \ddot{\lambda}\mathbf{z} \cdot \mathbf{z} = \ddot{\lambda}\left(\mathbf{z} \cdot \mathbf{z} + 1\right). \tag{19}$$

This expression is rearranged to obtain

$$\ddot{\lambda} = \frac{\mathbf{z}_2 \cdot \mathbf{z}}{\mathbf{z} \cdot \mathbf{z} + 1}. \tag{20}$$

Finally, equation (16) is used to retrieve the vector $\ddot{\mathbf{q}}$:

$$\ddot{\mathbf{q}} = \mathbf{z}_2 - \ddot{\lambda}\mathbf{z}. \tag{21}$$

4. Curve Derivatives of Order n

The derivative of order n can be derived from the derivatives up to order $n-1$ in much the same way as the second derivative is derived from the first. The first step is to approximate the nth derivative of $\mathcal{H}\left(c\left(s\right)\right)$. This is given by Faá de Bruno's formula [30]:

$$\frac{\mathrm{d}}{\mathrm{d}s}\mathcal{H}\left(c\left(s\right)\right) = \sum \frac{n!}{\prod_{j=1}^{n} j!^{m_j} m_j!} \nabla^{\sum_{j=1}^{n} m_j} \mathcal{H}\left(c\left(s\right)\right) \prod_{j=1}^{n} \left[c^{(j)}\left(s\right)\right]^{m_j}, \tag{22}$$

where the outer summation is taken over all n-tuples of non-negative integers $\{m_1, \ldots, m_n\}$ such that

$$\sum_{j=1}^{n} j m_j = n \tag{23}$$

and the notation which seems to indicate the product $\left[c^{(j)}\left(s\right)\right]^{m_j}$ for all j is intended to indicate that $c^{(j)}\left(s\right)$ appears with multiplicity m_j as input to $\nabla^{\sum_{j=1}^{n} m_j} \mathcal{H}\left(c\left(s\right)\right)$.

Since $\mathcal{H}\left(c\left(s\right)\right) = \mathbf{0}$, therefore we have $\frac{\mathrm{d}}{\mathrm{d}s}\mathcal{H}\left(c\left(s\right)\right) = \mathbf{0}$. Define $\mathbf{w}_n \in \mathbb{R}^N$ as

$$\mathbf{w}_n \equiv \frac{\mathrm{d}}{\mathrm{d}s}\mathcal{H}\left(c\left(s\right)\right) - \nabla\mathcal{H}\left(c\left(s\right)\right) c^{(n)}\left(s\right). \tag{24}$$

Then we obtain

$$\nabla\mathcal{H}\left(c\left(s\right)\right) c^{(n)}\left(s\right) = -\mathbf{w}_n. \tag{25}$$

Note that $\mathbf{w}_n$ is not a function of $c^{(n)}\left(s\right)$, so it is possible to approximate it using a generalized algorithm for the discrete directional derivative operators on $\left(\dot{c}\left(s\right), \ldots, c^{(n-1)}\left(s\right)\right)$. A framework for approximating directional derivatives of any order is described in Section 5.

Using equation (8), the under-determined system (25) can be compacted to the fully determined system

$$\nabla_{\mathbf{q}}\mathcal{H}\left(c\left(s\right)\right)\mathbf{z}_n = -\mathbf{w}_n, \tag{26}$$

where

$$\mathbf{z}_n = \mathbf{q}^{(n)} + \lambda^{(n)}\mathbf{z}, \tag{27}$$

$\mathbf{z}_n \in \mathbb{R}^N$. The linear system (26) can be solved numerically for $\mathbf{z}_n$.

The additional equation needed to solve for $\mathbf{q}^{(n)}$ and $\lambda^{(n)}$ comes from differentiating the arclength definition given by equation (4) $n-1$ times. This expression is obtained using the general Leibniz rule:

$$0 = \frac{\mathrm{d}^{n-1}}{\mathrm{d}s^{n-1}}\dot{c} \cdot \dot{c} = \sum_{k=0}^{n-1} \binom{n-1}{k} c^{(k+1)} \cdot c^{(n-k)}. \tag{28}$$

Algorithm 1: High-order curve derivative calculation with arclength parametrization for curve derivative of order n

Data: n, $\mathbf{q}$, λ, $\mathcal{H}(\mathbf{q}, \lambda)$, $\frac{\partial}{\partial \lambda}\mathcal{H}(\mathbf{q}, \lambda)$, $\nabla_{\mathbf{q}}\mathcal{H}(\mathbf{q}, \lambda)$
Result: $\dot{\mathbf{q}}(s), \ldots, \mathbf{q}^{(n)}(s), \dot{\lambda}(s), \ldots, \lambda^{(n)}(s)$
Calculate $\dot{c}(s)$
for $d = 2 : n$ **do**
  Calculate ϵ for $c^{(d-1)}(s)$
  Calculate $\mathbf{w}_n$ from equations (24) and (22)
  Solve $\nabla_{\mathbf{q}}\mathcal{H}\mathbf{z}_n = -\mathbf{w}_n$
  Calculate $\lambda^{(d)}(s)$ from equation (31)
  Calculate $\mathbf{q}^{(d)}(s)$ from equation (32)
end

Solving for $c^{(n)} \cdot \dot{c}$ gives

$$c^{(n)} \cdot \dot{c} = \begin{cases} -\sum_{k=1}^{\frac{n-3}{2}} \binom{n-1}{k} c^{(k+1)} \cdot c^{(n-k)} - \frac{1}{2}\binom{n-1}{\frac{n-1}{2}} c^{((n+1)/2)} \cdot c^{((n+1)/2)} & n \text{ is odd} \\ -\sum_{k=1}^{\frac{n}{2}-1} \binom{n-1}{k} c^{(k+1)} \cdot c^{(n-k)} & n \text{ is even,} \end{cases} \tag{29}$$

which can be evaluated numerically.

Taking the dot product of both sides of equation (27) with $\dot{\mathbf{q}}$ gives

$$\mathbf{z}_n \dot{\mathbf{q}} = \mathbf{q}^{(n)} \cdot \dot{\mathbf{q}} + \lambda^{(n)} \mathbf{z} \cdot \dot{\mathbf{q}} = c^{(n)} \cdot \dot{c} - \lambda^{(n)} \dot{\lambda} + \lambda^{(n)} \mathbf{z} \cdot \dot{\mathbf{q}}. \tag{30}$$

This can be rearranged and simplified using equations (11) and (12) to give:

$$\lambda^{(n)} = \frac{\mathbf{z}_n \cdot \dot{\mathbf{q}} - c^{(n)} \cdot \dot{c}}{\sqrt{\mathbf{z} \cdot \mathbf{z} + 1}}. \tag{31}$$

This is substituted into equation (27) to calculate $\mathbf{q}^{(n)}$:

$$\mathbf{q}^{(n)} = \mathbf{z}_n - \lambda^{(n)} \mathbf{z}. \tag{32}$$

The higher order derivative calculation is summarized as Algorithm 1. The calculation can alternatively be represented as an $N + 1$ by $N + 1$ system of equations:

$$\begin{pmatrix} \nabla_{\mathbf{q}}\mathcal{H}(c(s)) & \mathbf{z} \\ \dot{\mathbf{q}}(s) & \dot{\lambda}(s) \end{pmatrix} \begin{pmatrix} \mathbf{q}^{(n)}(s) \\ \lambda^{(n)}(s) \end{pmatrix} = \begin{pmatrix} -\mathbf{w}_n \\ x_n \end{pmatrix}, \tag{33}$$

where $x_n \in \mathbb{R}$ is calculated from the right-hand side of equation (29). Notice that the $N + 1$st column and $N + 1$st row are both dense. A procedure for solving a sparse linear system with a dense row and column appended is presented in Appendix A. This procedure involves two linear solves using the sparse sub-matrix $\nabla_{\mathbf{q}}\mathcal{H}(c(s))$, whereas the procedure presented in this section requires only one linear solve because it has been possible to recycle the solution to the linear solve from the tangent calculation.

5. Approximations to Directional Derivatives

While the Jacobian is usually represented by a matrix, it is more convenient in the current context to define the Jacobian by interpreting it as a directional derivative, as in the approach taken by Pönisch and Schwetlick [29].

Definition 1. *Let* $\mathbf{u} \in \mathbb{R}^{N_1}$, $\mathbf{v} \in \mathbb{R}^{N_1}$, *and let* $\mathcal{H} : \mathbb{R}^{N_1} \to \mathbb{R}^{N_2}$ *be at least* C^1 *differentiable. The* Jacobian *of* $\mathcal{H}(\mathbf{u})$ *is an operator* $\nabla\mathcal{H}(\mathbf{u}) : \mathbb{R}^{N_1} \to \mathbb{R}^{N_2}$ *such that*

$$(\nabla\mathcal{H}(\mathbf{u})\,\mathbf{v})_{[i]} = \sum_j \frac{\partial}{\partial \mathbf{u}_{[j]}} \mathcal{H}_{[i]}(\mathbf{u})\,\mathbf{v}_{[j]}. \tag{34}$$

where the notation $\mathbf{v}_{[j]}$ *indicates the* j*th element of the vector* $\mathbf{v}$.

Clearly the Jacobian is a linear operator with respect to $\mathbf{v}$. It can be seen from equation (34) that the mapping $\nabla\mathcal{H}(\mathbf{u})\,\mathbf{v}$ can be interpreted as the directional derivative of $\mathcal{H}(\mathbf{u})$ in the direction of $\mathbf{v}$ and with "speed" $\|\mathbf{v}\|$. Hence the mapping $\nabla\mathcal{H}(\mathbf{u})\,\mathbf{v}$ can be written in the Fréchet sense using forward-differencing:

$$\nabla\mathcal{H}(\mathbf{u})\,\mathbf{v} = \lim_{\epsilon\to 0} \frac{\mathcal{H}(\mathbf{u}+\epsilon\mathbf{v}) - \mathcal{H}(\mathbf{u})}{\epsilon}, \tag{35}$$

$\epsilon \in \mathbb{R}$, or centred-differencing:

$$\nabla\mathcal{H}(\mathbf{u})\,\mathbf{v} = \lim_{\epsilon\to 0} \frac{\mathcal{H}(\mathbf{u}+\epsilon\mathbf{v}) - \mathcal{H}(\mathbf{u}-\epsilon\mathbf{v})}{2\epsilon}. \tag{36}$$

Approximations to the directional derivatives can be formed using either equation (35) or equation (36) without the limit and choosing a finite value of ϵ. The parameter ϵ should be chosen to balance between the truncation error from the finite-differencing approximation and rounding error. Taking the approach of Nielsen et al. [27], the following expression is used:

$$\epsilon = \sqrt{\frac{N\delta}{\mathbf{v}^T\mathbf{v}}}, \tag{37}$$

where $\delta \in \mathbb{R}$ is typically taken around 10^{-12}, which is about 10^3 times machine precision when using double precision arithmetic. The intention in applying this formula is to acquire an ϵ such that the elements of $\epsilon\mathbf{v}$ are approximately the size of the corresponding elements of $\delta\mathbf{u}$. The vector $\mathbf{u}$ does not appear in the formula because it is assumed that the elements of $\mathbf{u}$ are of order unity. If this is not the case, then the elements of $\mathbf{v}$ can be divided by some benchmark value as appropriate. For example, if some elements of $\mathbf{u}$ are of order 10^3, the corresponding elements of $\mathbf{v}$ can be temporarily scaled by dividing by 10^3 before applying equation (37) and then scaled back to their original values.

The Hessian operator can also be interpreted as a directional derivative.

Definition 2. *Let* $\mathbf{u} \in \mathbb{R}^{N_1}$, $\mathbf{v}_1 \in \mathbb{R}^{N_1}$, $\mathbf{v}_2 \in \mathbb{R}^{N_1}$, *and let* $\mathcal{H} : \mathbb{R}^{N_1} \to \mathbb{R}^{N_2}$ *be at least* C^2 *differentiable. The* Hessian *of* $\mathcal{H}(\mathbf{u})$ *is an operator* $\nabla^2\mathcal{H}(\mathbf{u}) : \mathbb{R}^{N_1} \times \mathbb{R}^{N_1} \to \mathbb{R}^{N_2}$ *such that*

$$\left(\nabla^2\mathcal{H}(\mathbf{u})\,[\mathbf{v}_1,\mathbf{v}_2]\right)_{[i]} = \sum_k \sum_j \frac{\partial^2}{\partial \mathbf{u}_{[j]}\partial \mathbf{u}_{[k]}} \mathcal{H}_{[i]}(\mathbf{u})\,\mathbf{v}_{1[j]}\mathbf{v}_{2[k]}, \tag{38}$$

where the notation $\nabla^2\mathcal{H}(\mathbf{u})\,[\mathbf{v}_1,\mathbf{v}_2]$ *indicates that* $\nabla^2\mathcal{H}(\mathbf{u})$ *operates on the vector pair* $[\mathbf{v}_1,\mathbf{v}_2]$.

The Hessian is easily verified to be a bilinear operator with the property

$$\nabla^2 \mathcal{H}(\mathbf{u})[\mathbf{v}_1, \mathbf{v}_2] = \nabla^2 \mathcal{H}(\mathbf{u})[\mathbf{v}_2, \mathbf{v}_1]. \tag{39}$$

In addition, the following remark is easily verified by applying Definition 1 twice.

Remark 1. *Let* $\mathbf{u} \in \mathbb{R}^{N_1}$, $\mathbf{v}_1 \in \mathbb{R}^{N_1}$, $\mathbf{v}_2 \in \mathbb{R}^{N_1}$, *and let* $\mathcal{H} : \mathbb{R}^{N_1} \to \mathbb{R}^{N_2}$ *be at least* C^2 *differentiable. Then*

$$\nabla^2 \mathcal{H}(\mathbf{u})[\mathbf{v}_1, \mathbf{v}_2] = \nabla\left[\nabla \mathcal{H}(\mathbf{u})\mathbf{v}_1\right]\mathbf{v}_2. \tag{40}$$

Like the Jacobian, the Hessian also represents a directional derivative [29]; it can be interpreted as the directional derivative of $\mathcal{H}(\mathbf{u})$ in the direction of $\mathbf{v}_1$ differentiated for a second time in the direction of $\mathbf{v}_2$. As such, it can also be approximated by successively applying the Fréchet definition of the directional derivative. In accordance with Remark 1, the following approximation can be constructed by successively applying first-order approximations to the first derivative:

$$\nabla^2 \mathcal{H}(\mathbf{u})[\mathbf{v}_1, \mathbf{v}_2] \approx \frac{\mathcal{H}(\mathbf{u} + \epsilon_2 \mathbf{v}_2) - \mathcal{H}(\mathbf{u} - \epsilon_1 \mathbf{v}_1 + \epsilon_2 \mathbf{v}_2) - \mathcal{H}(\mathbf{u}) + \mathcal{H}(\mathbf{u} - \epsilon_1 \mathbf{v}_1)}{\epsilon_1 \epsilon_2}. \tag{41}$$

Similarly, the second-order approximation to the first derivative can be used:

$$\begin{aligned} &\nabla^2 \mathcal{H}(\mathbf{u})[\mathbf{v}_1, \mathbf{v}_2] \\ \approx &\frac{\mathcal{H}(\mathbf{u} + \epsilon_1 \mathbf{v}_1 + \epsilon_2 \mathbf{v}_2) - \mathcal{H}(\mathbf{u} - \epsilon_1 \mathbf{v}_1 + \epsilon_2 \mathbf{v}_2) - \mathcal{H}(\mathbf{u} + \epsilon_1 \mathbf{v}_1 - \epsilon_2 \mathbf{v}_2) + \mathcal{H}(\mathbf{u} - \epsilon_1 \mathbf{v}_1 - \epsilon_2 \mathbf{v}_2)}{2^2 \epsilon_1 \epsilon_2}. \end{aligned} \tag{42}$$

This method for calculating second-order directional derivatives can easily be extended to directional derivatives of any order. Algorithms 4 or 6 in Appendix B can be used to approximate directional derivatives of any order for the general case. If all directions of the directional derivative are the same (that is, all input vectors in the tensor-vector product are the same) then Algorithms 5 or 7, also in Appendix B, can be used to reduce the number of residual evaluations needed.

6. Curve Derivatives with λ Parametrization

The curve derivatives can also be calculated with a λ parametrization. Our convention is to define a λ parametrization implicitly by

$$\dot{\lambda}(r) = -1, \tag{43}$$

$r \in \mathcal{S}$, $\mathcal{S} \subset \mathbb{R}$, $R = \{[0, 1]\}$. Successively differentiating both sides of this equation yields the additional conditions:

$$\lambda^{(n)}(r) = 0 \tag{44}$$

for all $n > 1$. Differentiating $\mathcal{H}(c(r)) = \mathbf{0}$ and using condition (43) gives the expression for the first derivative:

$$\nabla_{\mathbf{q}} \mathcal{H}(c(r))\,\dot{\mathbf{q}}(r) = \frac{\partial}{\partial \lambda} \mathcal{H}(c(r)). \tag{45}$$

Note the useful property

$$\dot{\mathbf{q}}(r) = \sqrt{\mathbf{z} \cdot \mathbf{z} + 1}\, \dot{\mathbf{q}}(s) \tag{46}$$

which allows for easy conversion between $\dot{\mathbf{q}}(s)$ and $\dot{\mathbf{q}}(r)$.

The derivation for the higher derivatives for this parametrization proceeds in much the same way as the derivation for the higher derivatives with respect to the arclength parametrization. Differentiating $\mathcal{H}(c(r)) = \mathbf{0}$ n times gives an expression for $\frac{\mathrm{d}}{\mathrm{d}r}\mathcal{H}(c(r))$ analogous to equation (22). Define

$$\mathbf{w}'_n = \frac{\mathrm{d}}{\mathrm{d}r}\mathcal{H}(c(r)) - \nabla\mathcal{H}(c(r))\,\mathbf{q}^{(n)}(r), \tag{47}$$

$\mathbf{w}'_n \in \mathbb{R}^N$, where the prime distinguishes $\mathbf{w}'_n$ from $\mathbf{w}_n$. Since $\frac{\mathrm{d}}{\mathrm{d}r}\mathcal{H}(c(r)) = \mathbf{0}$, the expression for the nth derivative of $\mathbf{q}(r)$ is given by

$$\nabla\mathcal{H}(c(r))\,\mathbf{q}^{(n)}(r) = -\mathbf{w}'_n. \tag{48}$$

The $\mathbf{q}^{(n)}(r)$ calculation is summarized as Algorithm 2, where the vector $\mathbf{w}'_n$ can be evaluated by applying Algorithms 4 through 7 in Appendix B without modification.

Algorithm 2: High order curve derivative calculation with λ parametrization for curve derivative of order n

Data: n, $\mathbf{q}$, λ, $\mathcal{H}(\mathbf{q}, \lambda)$, $\frac{\partial}{\partial\lambda}\mathcal{H}(\mathbf{q}, \lambda)$, $\nabla_{\mathbf{q}}\mathcal{H}(\mathbf{q}, \lambda)$
Result: $\dot{\mathbf{q}}(r), \ldots, \mathbf{q}^{(n)}(r), \dot{\lambda}(r), \ldots, \lambda^{(n)}(r)$
Calculate $\dot{c}(r)$
for $d = 2 : n$ **do**
  Calculate ϵ for $c^{(d-1)}(r)$
  Calculate $\mathbf{w}'_n$ from equations (47) and (22)
  Solve $\nabla_{\mathbf{q}}\mathcal{H}\mathbf{q}^{(n)}(r) = -\mathbf{w}'_n$
end

7. Practical Considerations for Calculating High Derivatives of Curves

The primary cost in terms of CPU time is in forming the $\mathbf{w}_n$ vector and solving the linear system given by equation (25). A consideration which can reduce CPU time is that the matrix on the left-hand side of the equation is the same for any n. This can lead to efficiency improvement depending on the linear solver used. For example, an LU or ILU factorization need only be performed once and can be reused for the linear solve for each derivative.

The $\mathbf{w}_n$ calculation potentially requires numerous directional derivatives and their coefficients to be computed. The directional derivatives are identified and the coefficients are calculated using equation (22) with condition (23) and ignoring the $\nabla\mathcal{H}(c(s))\,c^{(n)}(s)$ term. While this is relatively straightforward, the complexity of the summation in equation (22) provides some challenges in terms of data allocation and establishing a logical indexing. The indexing system that we have developed comes from noticing that the sum of the orders of the derivatives is always equal to n and recalling that the input vectors to the directional derivatives commute (equation (39)). This means that the total number of directional derivatives needed to construct $\mathbf{w}_n$ is equal to

the number of different integer combinations which can be summed to make n, the order of the summands being irrelevant. This value, denoted $\mathcal{P}(n)$, is called the *partition function* and any non-ordered integer set whose sum equals n is called a *partition* [2]. The partition function can be evaluated using the recursive equation:

$$\mathcal{P}(n) = \sum_{k>0} (-1)^{k-1} \left[\mathcal{P}\left(n - g_k^-\right) + \mathcal{P}\left(n - g_k^+\right) \right], \tag{49}$$

$$g_k^- = \frac{k(3k-1)}{2},\ g_k^+ = \frac{k(3k+1)}{2},$$
$$\mathcal{P}(1) = 0,\ \mathcal{P}(0) = 0,\ \mathcal{P}(k) = 0 \text{ for } k < 0,$$
$$\mathcal{P} : \mathbb{Z} \to \mathbb{Z},\ g_k^- \in \mathbb{Z},\ g_k^+ \in \mathbb{Z},$$

where the summation is terminated when the condition $n > g_k^-$ is met.

The partitions are arranged anti-lexicographically using algorithm ZS1 of Zoghbi and Stojmenović [42], the parts of the partition representing the multiplicity of each derivative. As an example, the indexing matrix generated from the partitioning algorithm is shown paired with the coefficient vector generated from equation (22) for $n = 4$:

$$\begin{pmatrix} 4 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 2 & 2 & 0 & 0 \\ 2 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 4 \\ 3 \\ 6 \\ 1 \end{pmatrix}. \tag{50}$$

Since the order of the tensor for the jth term is equal to the number of non-zero entries in the jth row, this matrix and vector combination contains enough information to construct $\mathbf{w}_4$. Ignoring the top line, which corresponds to $\nabla\mathcal{H}(c(s))\, c^{(n)}(s)$, the expression for $\mathbf{w}_4$ is

$$\mathbf{w}_4 = 4\nabla^2\mathcal{H}[\dddot{c}, \dot{c}] + 3\nabla^2\mathcal{H}[\ddot{c}, \ddot{c}] + 6\nabla^3\mathcal{H}[\ddot{c}, \dot{c}, \dot{c}] + \nabla^4\mathcal{H}[\dot{c}, \dot{c}, \dot{c}, \dot{c}]. \tag{51}$$

The number of integers needed to be stored using the partition-based method is only $(n+1)\mathcal{P}(n)$, whereas a more naive scheme which allocates for every combination of $j \leq n$ would allocate for n^n. Table 1 shows that partition-based memory allocation and indexing becomes necessary as n becomes large.

n	5	10	15
$(n+1)\mathcal{P}(n)$	42	462	2816
n^n	3125	10^{10}	4.38×10^{17}

Table 1: Number of integer values stored for the partition-based storage scheme compared to a simple index-based storage scheme for calculating $\mathbf{w}_n$

While the number of residual evaluations required to evaluate equation (24) grows at a rapid rate, the cost is still quite low for practical values of n. Mackens [23] investigated a method for reducing this cost and also the complexity of the calculation. The method comes from the observation that

$$\left(\frac{\mathrm{d}}{\mathrm{d}s}\right)^n \mathcal{H}\left(c(s) + \sum_{j=1}^{n-1} \frac{\Delta s^j}{j!} c^{(j)}(s)\right)_{|\Delta s=0} = -\mathbf{w}_n. \tag{52}$$

The cost benefit of using equation (52) in place of equation (24) is that only a single directional derivative with a single direction must be computed for each derivative, resulting in much fewer residual evaluations.

As will be shown in Section 8.6, accuracy is currently of greater concern to the calculation than cost, and this method does not provide a solution to the accuracy problems encountered. Mackens [23] presented[2] error estimations for directional derivative calculations up to $n = 4$ using this method for a very low-dimensional homotopy in $\mathbb{R}^2$ and found that the numerical approximation error grew significantly as n was increased. Syam and Siyyam [36] performed a similar study and the same trend is seen in their second test case, their first case being a simple scalar case. Since the direction vectors in the directional derivatives can vary by 8 orders of magnitude or more for our applications it seems unlikely that the directional derivatives can be constructed with sufficient accuracy using this method.

The next topic that will be addressed is whether second-order or first-order first derivative approximations should be used to construct the tensor-vector products. From experimentation with both we have not observed any measurable accuracy improvement for the second-order approximation relative to the first-order approximation when calculating curve derivatives of up to the third derivative, at least not for ϵ values around 10^{-8}. However, some slight improvement seems apparent at higher derivatives. The cost measured in number of residual evaluations of forming the tensor-vector products for $\nabla^n \mathcal{H}$ is 2^n when using the second-order accurate method and $2^n - 1$ when using the first-order accurate method. Since the additional cost of using the second-order approximation is between modest and negligible, especially when considering the cost of solving the linear system, we opt to use the second-order accurate tensor-vector product estimates.

Another consideration which is important for efficiency is special consideration for the case where all direction vectors to the directional derivative are in the same direction. Though the generalized Algorithm 6 could be used for all directional derivatives, the number of residual evaluations needed to evaluate the tensor-vector product in this way is 2^n, whereas Algorithm 7, which takes advantage of all directions being the same, requires only $n + \mathrm{mod}\,(n, 2)$ residual evaluations. This is particularly significant when calculating the high curve derivatives because the highest order tensor-vector product appearing in the $\mathbf{w}_n$ calculation is always of this form.

The cost measured in number of residual evaluations needed to calculate $\mathbf{w}_n$ is shown in Table 2 for the various methods. The data in this table reinforce the recommendations made in this section. If high derivatives are desired then clearly there is cost benefit to Mackens' method if the accuracy issues can be overcome. There is also room for significant cost reduction if the tensor-vector product is fully generalized to account for any common direction vectors. For example, $\nabla^3 \mathcal{H}\,(c)\,[\ddot{c}, \dot{c}, \dot{c}]$ could be made more efficient by considering that two of the direction vectors are the same, reducing the cost from 8 residual evaluations to 6. While the cost reduction is modest for this example, it becomes very significant for higher order tensors and occurs often in the $\mathbf{w}_n$ calculation.

8. Validation

8.1. Flow Solver

The validation is performed using a Newton-Krylov-Schur parallel implicit flow solver based on a finite-difference [20] discretization applicable to multi-block structured grids. The finite-

[2] See the tables on pages 247, 248, and 250 of Mackens [23]

Method	n							
	2	3	4	5	6	7	8	9
1a	3	10	28	66	154	334	723	1515
1b	2	6	16	40	92	214	473	1009
2a	4	12	32	72	164	348	744	1544
2b	2	8	18	46	100	228	492	1038
Mackens	2	3	4	5	6	7	8	9

Table 2: Cost of evaluating $\mathbf{w}_n$ using various methods, where cost is measured in number of residual evaluations; the methods are:
1a) First-order accurate, general tensor-vector products only (Algorithm 4);
1b) First-order accurate, making use of single-direction directional derivatives (Algorithms 4 and 5);
2a) Second-order accurate, general tensor-vector products only (Algorithm 6);
2b) Second-order accurate, making use of single-direction directional derivatives (Algorithms 6 and 7)

difference discretization is based on the SBP-SAT [6, 9, 12, 16] approach, which uses Summation-By-Parts (SBP) operators to represent the discrete derivatives and Simultaneous Approximation Terms (SATs) to enforce the boundary conditions and couple the flow equations at block interfaces. The original inviscid flow solver flow of Hicken and Zingg [13] is used for all calculations in this paper though the flow solver has since been extended to viscous flows by Osusky and Zingg [28]. The flow solver can be used for both subsonic and transonic operating conditions. For transonic cases, a first-order dissipation operator is included with a pressure sensor [15] for shock capturing.

To parallelize the flow solver, the physical domain is decomposed into blocks. Parallel preconditioning of the Krylov solver is performed using the Schur complement method [32] with block incomplete lower-upper (ILU) preconditioning applied to the domain blocks. The specific type of ILU factorization used in the current study is known as ILU(p) [32], where p is the fill level. The ILU(p) factorization is constructed based on an approximate Jacobian matrix using nearest neighbour nodes only. Since the Schur preconditioner can vary slightly throughout the Krylov solution process, a flexible variant of the Krylov solver GMRES is used, which is termed Flexible Generalized Minimal Residual, or FGMRES [32].

8.2. Test Cases

The test cases presented in this paper are all inviscid external compressible aerodynamic flows. The discrete inviscid Euler equations which are being solved for these test cases are given, for example, by Hicken and Zingg [13]. In this context, the state vector $\mathbf{q}$ consists of four elements per node. For node index i:

$$\mathbf{q}_{[i]} = \begin{bmatrix} \rho_i \\ \rho_i u_i \\ \rho_i v_i \\ e_i \end{bmatrix}, \tag{53}$$

where ρ is the air density, u and v are Cartesian velocity components, a is the speed of sound, and e is the energy, which, under the ideal gas assumption, is related to pressure p and velocities u and v through the formula

$$p = (\gamma - 1)\left(e - \frac{1}{2}\rho\left(u^2 + v^2\right)\right), \tag{54}$$

where $\gamma \in \mathbb{R}$ is the heat capacity ratio and is taken as 1.4 for air.

The NACA 0012 geometry is a well-known two-dimensional airfoil geometry often used for CFD testing and benchmarking and for which a large amount of experimental data exists. See McCroskey [25] for a summary and assessment of experimental data collected for this geometry prior to 1987. The grid used to represent the region surrounding the NACA 0012 airfoil in our study has an H topology and consists of 15390 nodes divided evenly into 18 blocks.

8.3. Homotopy Continuation

The application considered is the convex homotopy [1]

$$\mathcal{H}(\mathbf{q}, \lambda) = (1 - \lambda)\mathcal{R}(\mathbf{q}) + \lambda\mathcal{G}(\mathbf{q}) = \mathbf{0}, \tag{55}$$

$$\mathcal{H} : \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}^N,\ \mathcal{G} : \mathbb{R}^N \to \mathbb{R}^N,\ \mathcal{R} : \mathbb{R}^N \to \mathbb{R}^N,\ \lambda \in \mathbb{R}.$$

This is developed into a homotopy continuation algorithm by considering the discrete form:

$$\mathcal{H}(\mathbf{q}, \lambda_k) = (1 - \lambda_k)\mathcal{R}(\mathbf{q}) + \lambda_k\mathcal{G}(\mathbf{q}) = \mathbf{0}, \tag{56}$$

$$k \in [0, m],\ \lambda_k \in \mathbb{R},\ \lambda_0 = 1,\ \lambda_m = 0,\ \lambda_{k+1} < \lambda_k.$$

Assume that equation (56) defines a regular homotopy between the solutions to $\mathcal{G}(\mathbf{q}) = \mathbf{0}$ and $\mathcal{R}(\mathbf{q}) = \mathbf{0}$. If $\mathcal{R}(\mathbf{q}) = \mathbf{0}$ is difficult to solve and $\mathcal{G}(\mathbf{q}) = \mathbf{0}$ is easy to solve, then tracing the curve defined implicitly by equation (56) with the conditions accompanying the equation leads to an approximation to the solution to $\mathcal{R}(\mathbf{q}) = \mathbf{0}$, which can be used as a starting point for a locally convergent method with a higher convergence rate, such as a quasi-Newton method. For the numerical studies in this paper, $\mathcal{R}$ is the discrete flow residual and $\mathcal{G}$ the second-difference dissipation operator with far-field boundary conditions used previously for homotopy continuation by Brown and Zingg [3, 5]. The curve can be traced numerically using either a predictor-corrector [1] or monolithic [4, 5] continuation algorithm.

8.4. Tangent

The validation of the tangent vector calculation is performed for inviscid flow over the two-dimensional NACA 0012 airfoil at Mach 0.3 and an angle of attack of 1°. Since the tangent vector cannot be calculated analytically, finite-differencing is used for the validation. The solution is needed at two consecutive points along the curve, which are obtained using the predictor-corrector method [1, 3]. The change in the arclength Δs is estimated by taking the standard norm of the difference in solution between these points:

$$\Delta s_i \approx \Delta s_i^* = \sqrt{(\mathbf{q}_{i+1} - \mathbf{q}_i) \cdot (\mathbf{q}_{i+1} - \mathbf{q}_i) + (\lambda_{i+1} - \lambda_i)^2}. \tag{57}$$

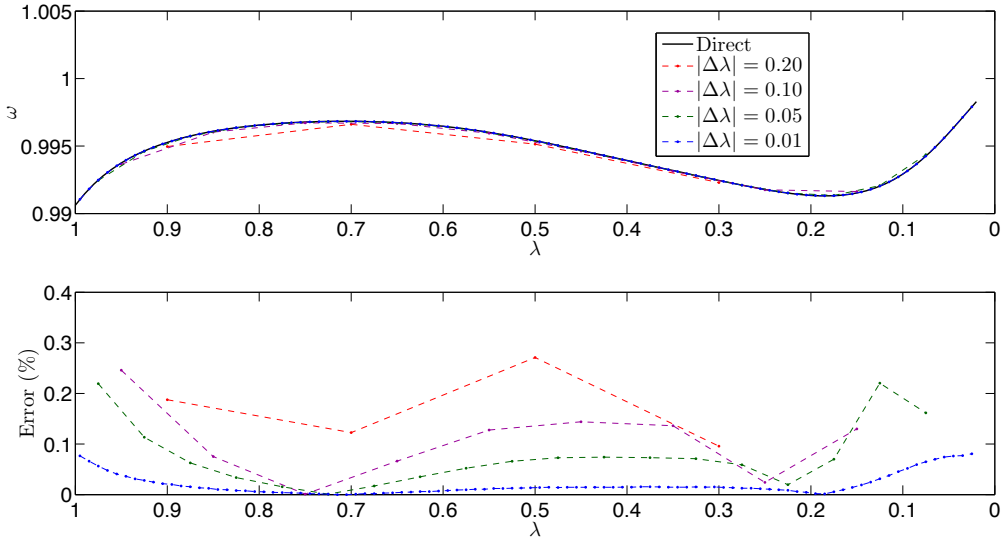


Figure 1: Comparison of ω calculated using the direct and finite-difference (FD) method with varying step size $|\Delta\lambda|$ on the inviscid NACA 0012 airfoil at Mach 0.3 and an angle of attack of 1°

A centred-difference estimate of the tangent vector is constructed by the equation

$$\dot{c}\left(s_{i+\frac{1}{2}}\right) \approx \dot{c}^*\left(s_{i+\frac{1}{2}}\right) = \frac{1}{\Delta s_i^*}\left(c\left(s_{i+1}\right) - c\left(s_i\right)\right). \tag{58}$$

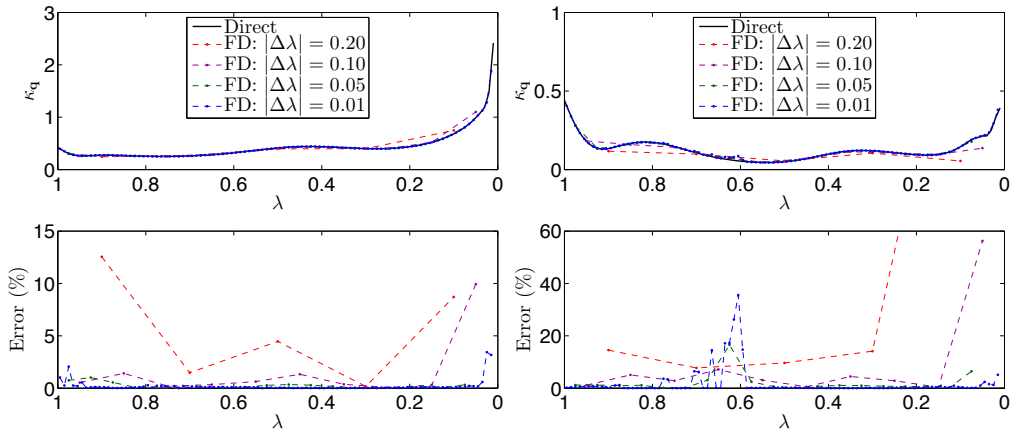
Since $\dot{c}(s) \cdot \dot{c}(s) = 1$, this check is analogous to comparing $\left|\dot{\lambda}(s)\right|$ using the direct or finite-differencing method.

The validation is performed by determining whether the finite-difference estimates of $\omega \equiv \sqrt{\dot{\mathbf{q}}(s) \cdot \dot{\mathbf{q}}(s)}$ converge to the ω values obtained from the direct method in the limit of the finite-difference step size $|\Delta\lambda|$ going to 0. The finite-difference approximation is interpreted as representing the derivative at $s_{i+\frac{1}{2}} \approx \frac{1}{2}(s_{i+1} + s_i)$, so when the error is calculated the values of ω calculated from the direct method are interpolated to these points.

To ensure that the effects of numerical error are not prevalent in the direct calculation, the points on the curve are solved for quite accurately by enforcing an absolute tolerance of $\|\mathcal{H}(c(s))\| < 10^{-8}$ in the corrector phase of the predictor-corrector method and by solving the linear systems appearing in the tangent calculation to a relative tolerance of 10^{-8} using a flexible variant of GCROT [8], denoted GCROT(m,k) [14]. However, forward-differencing is used to form all Jacobian-vector products in the linear solver, so that the effective tolerance to which the linear systems are solved is actually larger than 10^{-8}. The comparison is shown in Figure 1, which demonstrates that the finite-difference estimates do converge to the direct tangent estimate in the limit of $|\Delta\lambda|$ vanishing.

8.5. Curvature

As with the validation for the tangent calculation, the test case is inviscid flow over the two-dimensional NACA 0012 airfoil on the same grid. Two cases are investigated: the first is a



(a) Subsonic: Mach 0.3, angle of attack 1°

(b) Transonic: Mach 0.8, angle of attack 3°

Figure 2: Comparison of $\kappa_{\mathbf{q}}$ calculated using the direct and finite-difference (FD) method with varying step size $|\Delta\lambda|$ on the inviscid NACA 0012 airfoil at subsonic and transonic conditions

subsonic case at Mach 0.3 and an angle of attack of 1°, the second is a transonic case at Mach 0.8 and an angle of attack of 3°.

The backwards-difference estimate of the second derivative is obtained by dividing the difference in the tangent vector calculated at the current point and immediately previous point along the curve by Δs^*:

$$\ddot{c}\left(s_{i+\frac{1}{2}}\right) \approx \ddot{c}^*\left(s_{i+\frac{1}{2}}\right) = \frac{1}{\Delta s_i^*}\left(\dot{c}\left(s_{i+1}\right) - \dot{c}\left(s_i\right)\right). \tag{59}$$

The validation is performed by determining whether the finite-difference estimates of $\kappa_{\mathbf{q}}$ converge to the $\kappa_{\mathbf{q}}$ values obtained from the direct method in the limit of the finite-difference step size $|\Delta\lambda|$ going to 0. From Figure 2, it is apparently the case. To explain the smaller $\kappa_{\mathbf{q}}$ values for the transonic case, it is because the arclength parametrization causes $\kappa_{\mathbf{q}}$ to be proportional to $1/\sqrt{\mathbf{q}\cdot\mathbf{q}}$ and this case is at a higher Mach number.

Note that once the finite-difference step size $|\Delta\lambda|$ becomes sufficiently small, the error in the finite-difference approximation to $\kappa_{\mathbf{q}}$ will begin to increase with decreasing step size. This can be explained by considering the error vectors $\mathbf{e}\left(s_i\right), \mathbf{e}\left(s_{i+1}\right) \in \mathbb{R}^N$ associated with the tangent vectors estimated from the direct method. These error vectors are independent of the estimated step size $\Delta s^* = s_{i+1} - s_i$. Then the curvature approximation can be written as

$$\ddot{c}^*\left(s\right) = \frac{1}{\Delta s^*}\left(\dot{c}\left(s_{i+1}\right) + \mathbf{e}\left(s_{i+1}\right) - \dot{c}\left(s_i\right) - \mathbf{e}\left(s_i\right)\right). \tag{60}$$

As Δs^* goes to 0, $\frac{1}{\Delta s^*}\left(\dot{c}\left(s_{i+1}\right) - \dot{c}\left(s_i\right)\right)$ approaches $\ddot{c}\left(s\right)$ but, since $\mathbf{e}\left(s\right)$ does not decrease with Δs, $\frac{1}{\Delta s^*}\left|\mathbf{e}\left(s_{i+1}\right) - \mathbf{e}\left(s_i\right)\right|$ will grow as Δs^* decreases and will eventually dominate the calculation.

8.6. Higher Derivatives

The test case is again inviscid flow over the two-dimensional NACA 0012 airfoil at Mach 0.3 and angle of attack of 1°. The backwards-difference estimate of the nth derivative is obtained

using the $n-1$st derivative:

$$c^{(n)}\left(s_{i+\frac{1}{2}}\right) \approx c^{(n)*}\left(s_{i+\frac{1}{2}}\right) = \frac{1}{\Delta s_i^*}\left(c^{(n-1)}\left(s_{i+1}\right) - c^{(n-1)}\left(s_i\right)\right). \tag{61}$$

Defining

$$\kappa_{\mathbf{q}}^{(n)} \equiv \sqrt{\mathbf{q}^{(n)} \cdot \mathbf{q}^{(n)}}, \tag{62}$$

$\kappa_{\mathbf{q}}^{(n)} \in \mathbb{R}$, the validation is performed by determining whether the backwards-difference estimates of $\kappa_{\mathbf{q}}^{(n)}$ converge to the $\kappa_{\mathbf{q}}^{(n)}$ values obtained from the direct method in the limit of the finite-difference step size $|\Delta\lambda|$ going to 0.

It is apparent from Figures 3a, c, and e that the higher order derivative calculations have been implemented correctly. However, it is also observed that the calculation is sensitive to the value of δ. Figure 3 shows the $\kappa_{\mathbf{q}}^{(n)}$ values for both $\delta = 10^{-6}$ and $\delta = 10^{-8}$, where δ refers to the δ value used in the $\mathbf{w}_n$ calculations only; $\delta = 10^{-12}$ is used for the matrix-vector products in all linear solves in both cases.

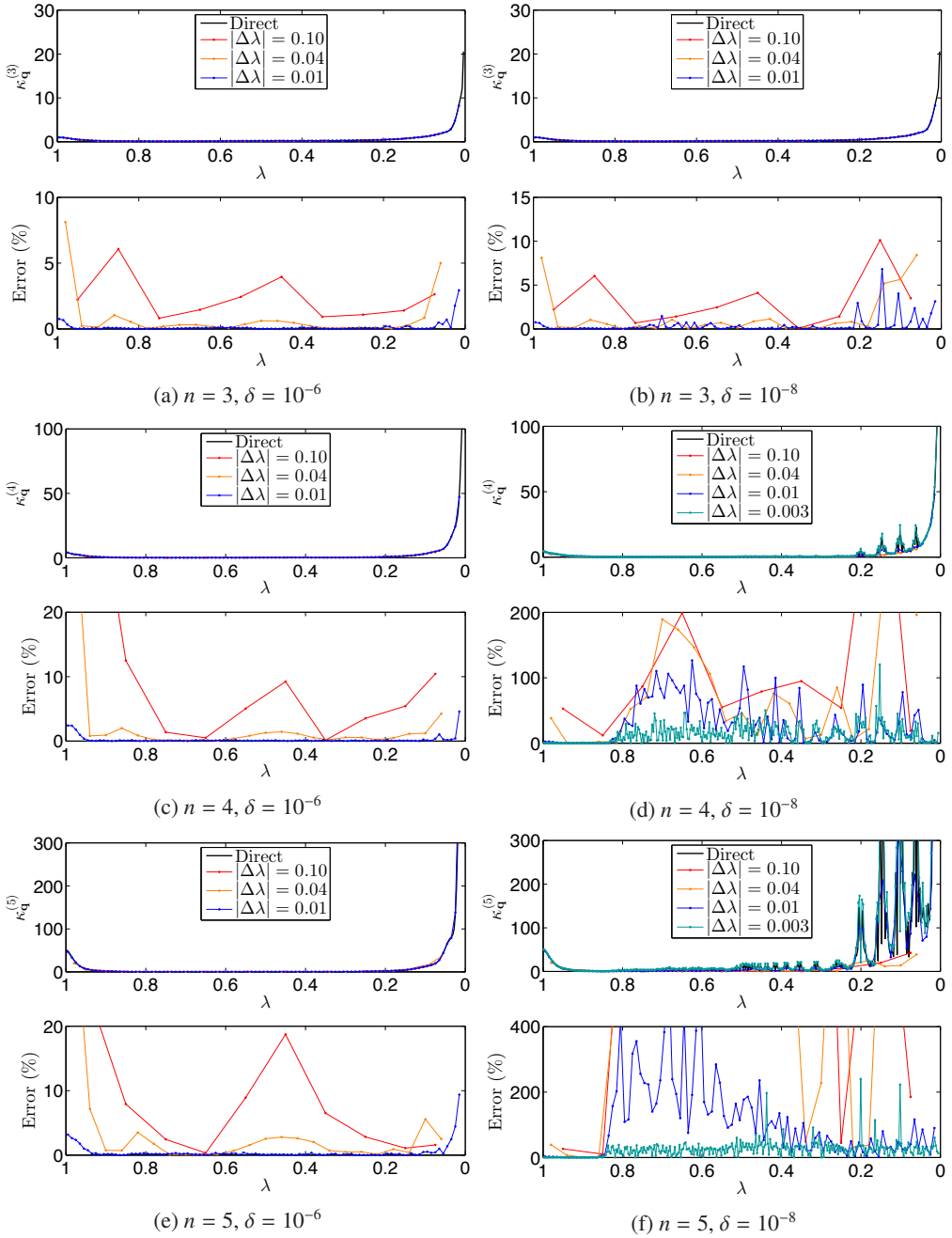
It is clear that the accuracy of the $\kappa_{\mathbf{q}}^{(n)}$ calculations is sensitive to δ. When estimating directional derivatives with finite-difference approximations such as equation (42), it is expected that the estimate will be accurate for a certain range of δ. When δ is too large, truncation error will dominate and when δ is too small, rounding error will dominate. When plotting error versus δ, a "V" pattern is thus expected. One might infer then from Figure 3, for which the $\kappa_{\mathbf{q}}^{(n)}$ calculation has been performed using double precision arithmetic, that $\delta = 10^{-8}$ is too small and that rounding error is dominating the calculation for this value of δ. However, when increasing the arithmetic precision from double precision (64 bit) to quadruple precision (128 bit), this is revealed not to be the case. A comparison of the error in the curvature calculation using double and quadruple precision arithmetic for several values of δ is shown in Figure 4. It is apparent from the figure that the higher derivative calculations for δ values as small as 10^{-12} are in very close agreement for the same calculation performed in double and quadruple precision, indicating that rounding error for the double precision calculations is not very significant for these values of δ. While the calculation seems accurate for $\delta = 10^{-6}$, it becomes inaccurate for smaller δ in the range $10^{-8} \geq \delta \geq 10^{-12}$. As δ is decreased beyond 10^{-12}, the accuracy of the calculation improves for the quadruple precision calculation but rounding error begins to dominate for the double precision calculation. For the quadruple precision calculation, the error eventually begins to increase again around 10^{-20}.

We have observed so far that the curvature calculation is sensitive to the value of δ used in the tensor-vector product estimation. We now investigate whether the tensor-vector product error is high or is being amplified in the curvature calculation. To perform this investigation, the relative error is estimated for $\mathbf{w}_n$ for each δ by comparing to the $\mathbf{w}_n$ corresponding to the value of δ which appears to give the most accurate $\kappa_{\mathbf{q}}$. For $n = 3$ and $n = 4$, $\delta = 10^{-18}$ corresponds to the benchmark value of $\mathbf{w}_n$ and for $n = 5$, $\delta = 10^{-16}$ is used. Quadruple precision arithmetic is used for the calculations.

For consistency with the $\kappa_{\mathbf{q}}$ error estimates, the relative error in $\mathbf{w}_n$ is estimated according to the following formula:

$$e_{\mathbf{w}_n} \approx \frac{\sqrt{\mathbf{w}_n \cdot \mathbf{w}_n} - \sqrt{\mathbf{w}_{n,\mathrm{b}} \cdot \mathbf{w}_{n,\mathrm{b}}}}{\sqrt{\mathbf{w}_{n,\mathrm{b}} \cdot \mathbf{w}_{n,\mathrm{b}}}}, \tag{63}$$

where $\mathbf{w}_{n,\mathrm{b}}$ is the benchmark value of $\mathbf{w}_n$. The error estimate for $\kappa_{\mathbf{q}}^{(n)}$ is compared to the error estimate for $\mathbf{w}_n$ in Figure 4, from which we see that the two quantities are correlated. This



(a) $n = 3$, $\delta = 10^{-6}$

(b) $n = 3$, $\delta = 10^{-8}$

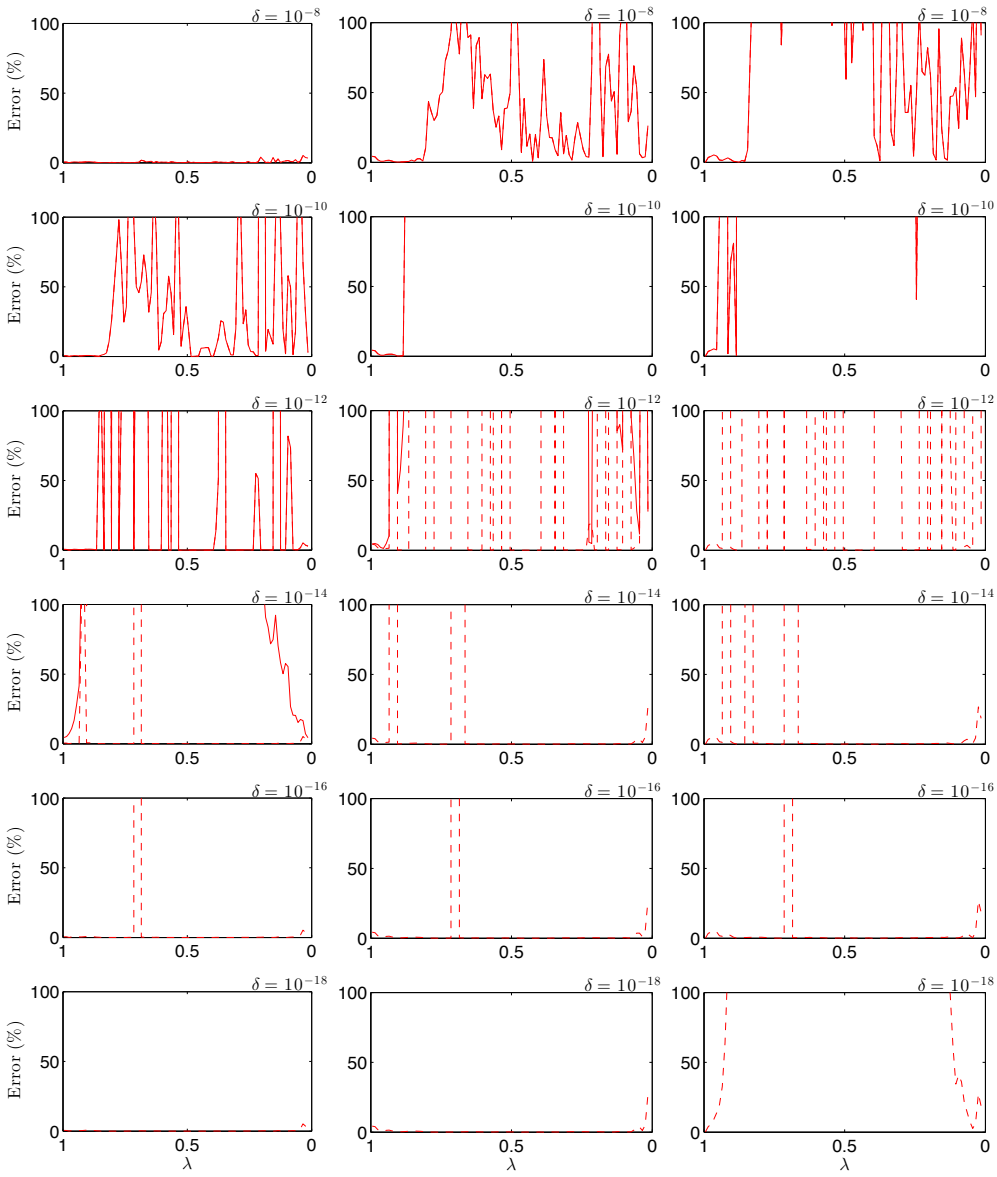
(c) $n = 4$, $\delta = 10^{-6}$

(d) $n = 4$, $\delta = 10^{-8}$

(e) $n = 5$, $\delta = 10^{-6}$

(f) $n = 5$, $\delta = 10^{-8}$

Figure 3: Comparison of $\kappa_{\mathbf{q}}^{(n)}$ for $n = 3$, $n = 4$, and $n = 5$ calculated using the direct and finite-difference (FD) method with different step size $|\Delta\lambda|$ on the inviscid NACA 0012 airfoil at Mach 0.3 and angle of attack of 1°



(a) $n = 3$ (b) $n = 4$ (c) $n = 5$

Figure 4: Comparison of the error in the direct and finite-difference estimates of $\kappa_{\mathbf{q}}^{(n)}$ calculated using double precision (solid line) and quadruple precision (dashed line) at $|\Delta\lambda| = 0.01$ for an inviscid NACA 0012 airfoil at Mach 0.3 and angle of attack of 1°; if only the solid line is visible it is because the two lines overlap; otherwise, if a line is not visible it is because the error is in excess of 100%

observation indicates that the high error which can appear in the curvature calculations can only be reduced by reducing the error in the tensor-vector product estimates.

One method to accurately estimate Jacobian-vector products is the complex step method [22, 24, 26, 35] because subtractive cancellation errors can be avoided. However, this method loses its usefulness when applied to tensor-vector products because subtractive cancellation errors are reintroduced at higher derivatives. It may be possible to instead use hyper-dual numbers [11] or hyper-complex numbers. This is recommended as a future research direction.

9. Higher Order Predictors

The development of higher order derivatives for implicitly-defined curves has applications to homotopy continuation. There are several ways in which higher order predictors can be constructed for implicitly-defined curves. It is possible to use polynomial extrapolation [1] such as Newton or Lagrange extrapolation, which use previous solution points, or Hermite extrapolation, which additionally requires previous tangent calculations, and can include higher derivatives as well. Lundberg and Poore [21] have advocated the use of a variable-order Adams-Bashforth method for certain applications. Another class of predictors which have appeared in the literature, most commonly in the field of computational structural mechanics but also for simple fluid mechanics problems, are known as asymptotic numerical methods [7, 18, 37]. This formulation requires that the system $\mathcal{H}$ can be written in terms of polynomial operators such as linear, quadratic, and possibly higher order, which poses a restriction on the problems to which asymptotic numerical methods can be applied.

9.1. Predictors Based on Taylor Polynomials

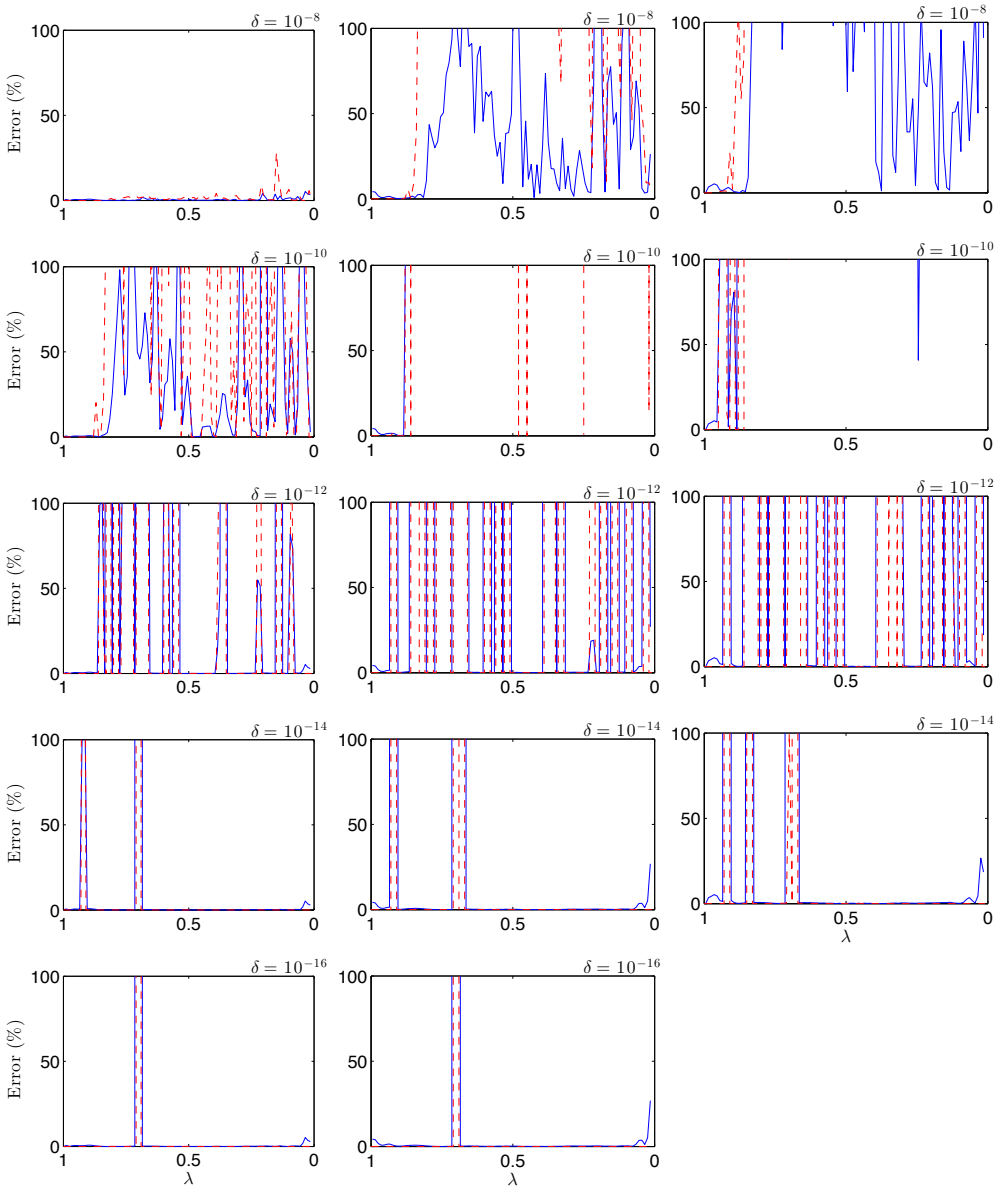
The higher order predictors investigated in this study are those built on Taylor polynomials. Since these predictors use only current information, they are generally expected to be more accurate and more stable than polynomial extrapolation or Adams-Bashforth methods, or Hermite extrapolation if limited to the first derivative, and, unlike asymptotic numerical methods, they are applicable to any continuous regular implicitly-defined curve. The disadvantage is the higher computational cost due to the higher derivatives which must be constructed. The higher order predictors studied in this section are constructed using the Taylor polynomial

$$c\left(s+\Delta s\right) = c\left(s\right) + \sum_{j=1}^{n} \frac{\left(\Delta s\right)^{j}}{j!} c^{(j)}\left(s\right) \tag{64}$$

centred at the current approximation to $c\left(s\right)$, presented here with arclength parametrization.

Parametrizing by arclength is convenient for applying steplength adaptation but introduces additional implementation complexity when relating Δs to $\Delta\lambda$, which must be performed when adjusting $\Delta\lambda$ based on user-imposed minimum and maximum values or when the $\lambda = 0$ condition has been reached. Calculating Δs for a given $\Delta\lambda$ can be performed numerically by applying the bisection method to equation (64).

The bisection method is a bracketing method requiring for an interval to be specified in which a solution is known to exist. To get left and right endpoints of this interval, we start by using



(a) $n = 3$ (b) $n = 4$ (c) $n = 5$

Figure 5: Error estimate for $\mathbf{w}_n$ (dashed line) compared to the error estimate for $\kappa_{\mathbf{q}}^{(n)}$ (solid line)

equation (64) with $n = 1$ to get an approximation to the arclength:

$$\Delta s \approx \Delta s^* = \frac{\|c(s + \Delta s) - c(s)\|}{\dot{c}(s)}. \tag{65}$$

We then check if the value of λ produced by equation (64) is greater or less than the target value of λ and set it to either the right or left endpoint of the interval for the bisection method. The other side of the interval is determined by incrementally increasing or decreasing Δs, as appropriate, until equation (64) gives a value for λ which is either less or greater, as appropriate, than the target value. Since we are solving for the roots of a higher order polynomial, it is possible that multiple $\Delta\lambda$ exist for a given Δs or that no $\Delta\lambda$ exist, in which case this procedure will fail. In such a case, the order of the polynomial can be reduced and the process repeated. The details of the calculation are presented in Algorithm 8 in Appendix B.

9.2. Accuracy of High Order Predictors

The effectiveness of the higher order predictor is investigated by considering the convex homotopy with dissipation operator applied to inviscid flow over the two-dimensional NACA 0012 airfoil at Mach 0.3 and angle of attack of 1° using the same grid as was used in the validation studies. Derivatives of up to order 9 are generated at $\lambda = 0.9$, 0.7, and 0.4, and the residual is calculated along the trajectory predicted by the Taylor polynomial. To ensure that the derivatives are calculated as accurately as possible, GCROT(m,k) is used to solve the linear systems which appear in the tangent and curvature calculations to a relative tolerance of 10^{-10}. However, the relative residual at the end of the linear solve is not actually this low due to the use of finite-difference estimates to form the Jacobian-vector products in the linear solvers.

As shown in Figure 6a, the higher order predictors predict the curve extremely well in a small radius Δs around s_0, usually corresponding to $|\Delta\lambda|$ values between 0.1 and 0.2, but the benefit of the higher order predictor quickly diminishes outside of this radius. When the predictor step is large, the higher order predictor can often be seen to give a worse prediction than even the tangent ($n = 1$) predictor. This is expected behaviour for a Taylor polynomial since the approximation is only valid within some radius. However, the various sources of rounding and truncation error from the derivative calculations may also be inhibiting performance.

A second study has also been performed to investigate how sensitive the predictor is to error. This study is analogous to the previous one, but the linear systems were only solved to a relative residual of 10^{-2}. The predicted trajectory is compared to that of the previous study in Figure 6b. This study is necessary for assessing the viability of the predictor as part of a continuation algorithm since solving each linear system to a relative residual of 10^{-10} is too expensive to be used in a cost-competitive algorithm.

While $\|\mathcal{H}(\mathbf{q}, \lambda)\| < 10^{-6}$ can no longer be maintained near s_0 when the accuracy of the derivative approximations is relaxed, the curve is still predicted quite well. Comparing the predictions of the lift coefficient C_l and drag coefficient C_d surrogate curves, also shown in Figure 6b, the performance degradation of the predictor when relaxing the accuracy of the curve derivatives is not as severe as it would appear when only considering the residual. While being able to maintain $\|\mathcal{H}(\mathbf{q}, \lambda)\| < 10^{-6}$ along the predicted trajectory is remarkable, this level of accuracy is unnecessary for effective continuation, and the high-order polynomial predictor is still effective when the linear solver tolerance is relaxed.

It is apparent from the C_l and C_d values calculated along the predicted trajectories as shown

in Figure 6b that the curve can be predicted reasonably well within a larger radius with higher order predictors than with a tangent predictor and at a reasonable cost increase. However, the radius for which the higher order predictor is valid remains unpredictable. When considering the suitability of these predictors as part of a continuation algorithm, robustness and accuracy are both of concern at the current state of development.

9.3. Performance Investigation for Predictors Based on Second-Degree Polynomials

We conduct only a preliminary investigation of the cost-effectiveness of incorporating higher derivatives into a curve-tracing algorithm, since a proper assessment would require many test cases and should include consideration for step-length adaptation, which goes beyond the scope of this paper. To conduct our comparison, we select the subsonic NACA 0012 case previously described and fix the step size at $\Delta\lambda = -0.05$. To reduce dependence of the results on the computational hardware, timing data is presented using TauBench [10] work units (wu), which is the total wall time divided by one TauBench unit. The TauBench program emulates the time taken by the DLR Tau code under certain input parameters and one TauBench unit is the time taken to run the TauBench codes under these parameters. The benchmark that was used in this paper is 2.50×10^5 nodes on 1 processor with 10 iterative steps, from which we have measured one TauBench unit as 9.571s on the SciNet general purpose cluster where the cases were run.

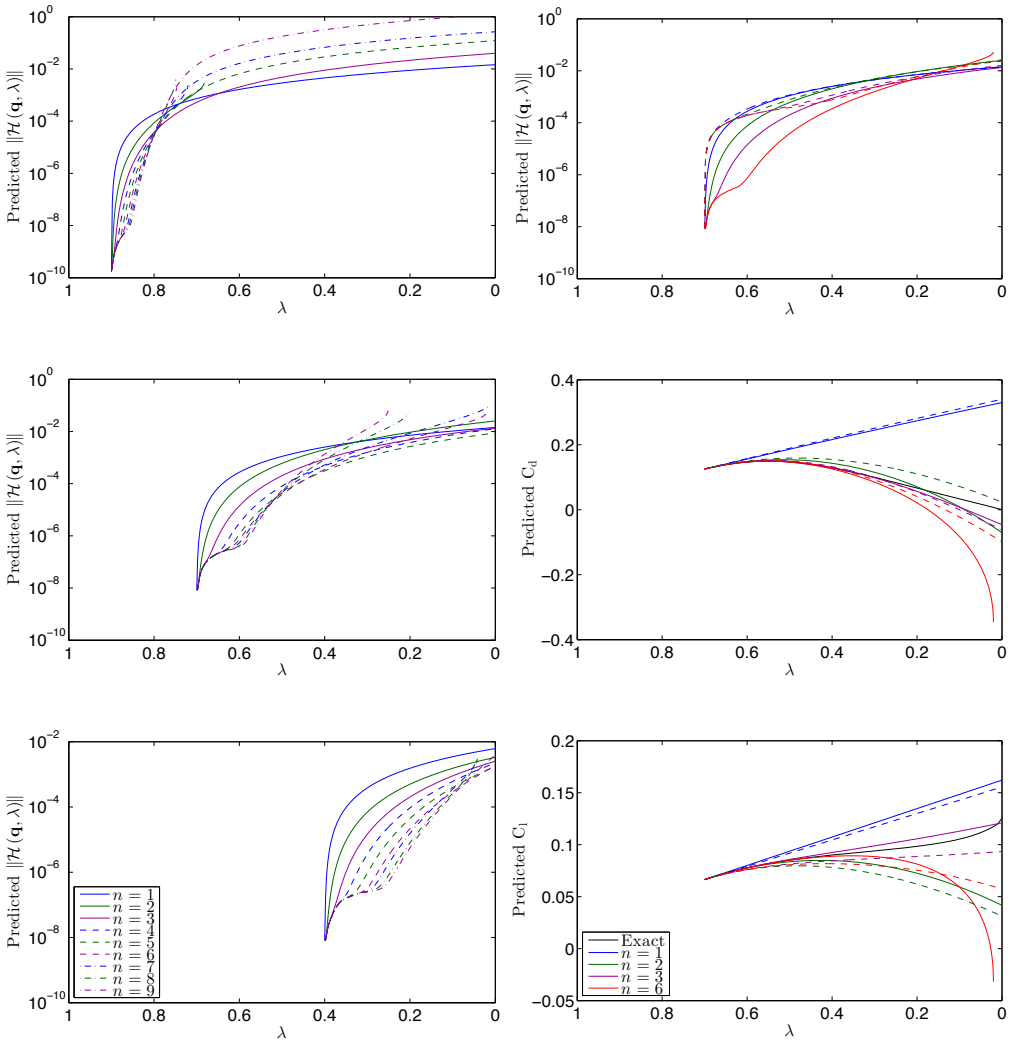
The performance when including the curvature vector in the predictor is compared to the performance when predictor information is limited to the tangent vector. For both cases, the corrector phase is terminated when the residual $\|\mathcal{H}(\mathbf{q}, \lambda)\|$ drops one order of magnitude relative to the embedded value[3]. The linear systems are solved two orders of magnitude using a Schur-block ILU(p) preconditioner with $p = 1$.

The performance of the two algorithms is compared using several figures. Figure 7 shows the algorithm progress as a function of time, expressed in TauBench work units, from which we see that the algorithm using the second-derivative lags behind the tangent-predictor algorithm slightly throughout. Figure 8 shows why this is the case. The second-derivative predictor gives a better prediction than the tangent predictor, which is apparent from the lower starting residual at each corrector phase. Though the improvement does not appear significant on the logarithmic scale of the plot, it is substantial enough that the corrector phase is generally completed in 4 iterations rather than 3. As shown by Figure 9, the cost benefit of this is essentially nullified by the additional cost incurred from the additional linear solve required for the curvature calculation. However, having a better initial iterate in the corrector phase can also improve the robustness, a benefit which is not quantified by this test case since both test cases converged.

10. Conclusions

A method for calculating the higher derivatives of regular implicitly defined curves has been developed which is suitable for application to curves defined implicitly by large sparse systems of nonlinear equations. The method was validated using a parallel implicit CFD flow solver and

[3]The embedded value of $\mathcal{H}$ is the predicted $\mathcal{H}$ where λ is updated but $\mathbf{q}$ is copied over from the previous corrector phase without predictor



(a) $\tau_l = 10^{-10}$ (b) $\tau_l = 10^{-10}$ (solid) and $\tau_l = 10^{-2}$ (dashed)

Figure 6: Residual at the predicted state from Taylor polynomials of order n calculated at $\lambda = 0.9$, 0.7, and 0.4 for the inviscid subsonic NACA 0012 case; the effect of the linear solver tolerance τ_l is investigated

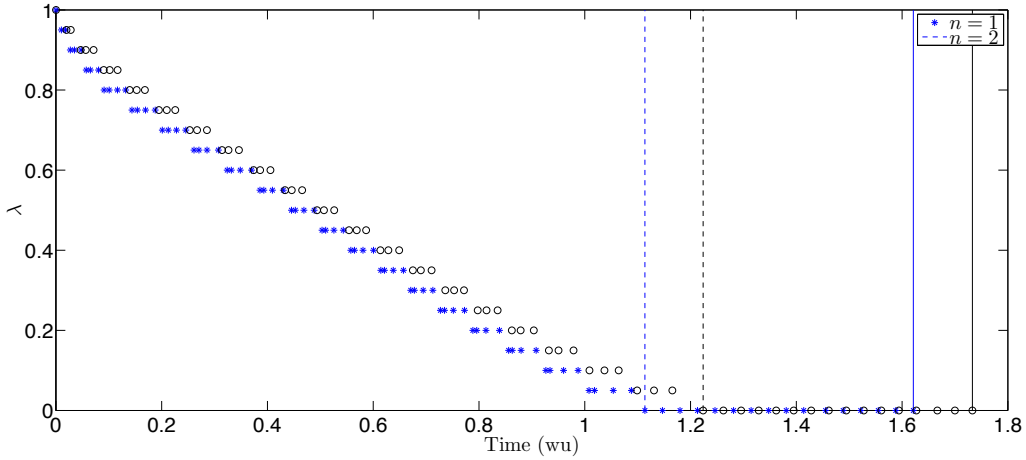


Figure 7: Convergence progress using two predictors; the dashed line indicates that globalization has been achieved and the solid line indicates that convergence has been achieved

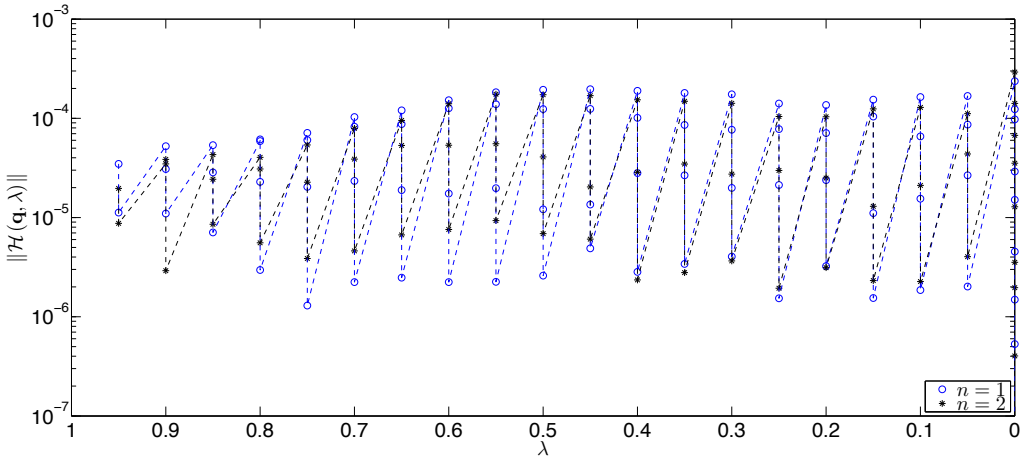


Figure 8: Tracking residual history using two predictors; the number of corrector iterations taken in each corrector phase is also visible

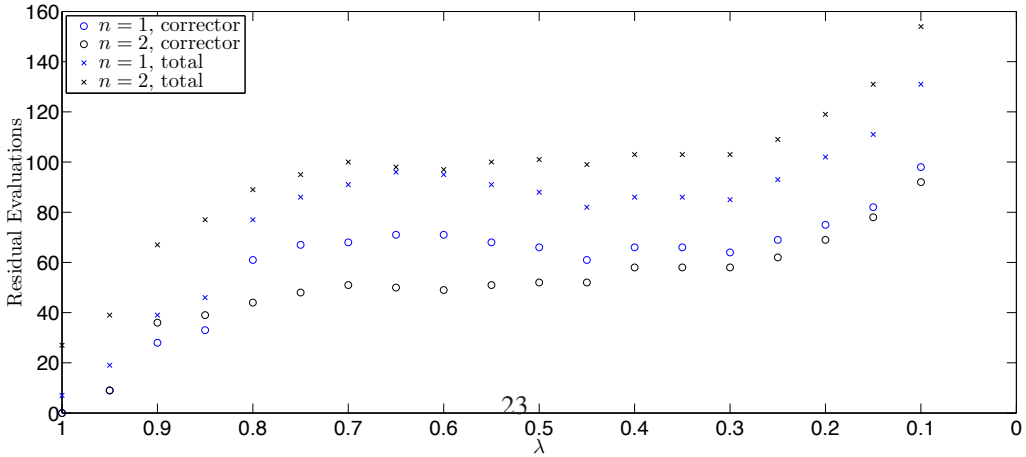


Figure 9: Number of residual evaluations taken at each value of λ; the 'o' symbols indicate the residual evaluations from corrector iterations only and the 'x' symbols indicate total residual evaluations from both corrector and predictor phases

was found to be efficient and exhibited the expected accuracy trend when compared with a finite-difference approximation. However, the method used to approximate the tensor-vector products required for part of the calculation was found to be unreliable and further work should be aimed at developing new methods to approximate these quantities reliably using double precision arithmetic and without forming the tensors explicitly.

A potential application of the methodology is the construction of accurate predictors for homotopy continuation algorithms. A preliminary study was performed for a homotopy of practical interest to computational aerodynamics. Two algorithms were compared, which were identical except that one used a tangent predictor and the other used a predictor based on a second-degree Taylor polynomial. It was observed that similar convergence time but superior curve-tracing accuracy could be achieved using the predictor based on the second-degree polynomial.

Acknowledgments

The authors gratefully acknowledge financial assistance from the Natural Science and Engineering Research Council (NSERC), the Canada Research Chairs program, and the University of Toronto. Computations were performed on the GPC supercomputer at the SciNet HPC Consortium. SciNet is funded by: the Canada Foundation for Innovation under the auspices of Compute Canada; the Government of Ontario; Ontario Research Fund - Research Excellence; and the University of Toronto. The authors also acknowledge the contribution of SciNet applications analyst Dr. Scott Northrup who assisted in some of the technical aspects of the quadruple precision calculations.

References

[1] Allgower, E. L., Georg, K., 1990. Introduction to Numerical Continuation Methods. Society for Industrial and Applied Mathematics.

[2] Andrews, G. E., 1984. The Theory of Partitions. Cambridge University Press.

[3] Brown, D. A., Zingg, D. W., June 2013. Advances in homotopy-based globalization strategies in computational fluid dynamics. AIAA-2013-2944.

[4] Brown, D. A., Zingg, D. W., July 2014. A new monolithic homotopy continuation algorithm with CFD applications. In: Eighth International Conference on Computational Fluid Dynamics. Chengdu, China.

[5] Brown, D. A., Zingg, D. W., 2016. A monolithic homotopy continuation algorithm with application to computational fluid dynamics. J. Comp. Phys. Submitted.

[6] Carpenter, M. H., Gottlieb, D., Abarbanel, S., 1994. Time-stable boundary conditions for finite-difference schemes solving hyperbolic systems: methodology and application to high-order compact schemes. J. Comput. Phys. 111 (2), 220–236.

[7] Cochelin, B., 1994. A path-following technique via an asymptotic-numerical method. Comput. Struct. 53 (3), 1181–1192.

[8] de Sturler, E., 1999. Truncation strategies for Krylov subspace methods. SIAM J. Numer. Anal. 36, 864–889.

[9] Del Rey Fernández, D. C., Hicken, J. E., Zingg, D. W., 2014. Review of summation-by-parts operators with simultaneous approximation terms for the numerical solution of partial differential equations. Comput. Fluids 95, 171–196.

[10] DLR Germany, 2014. Taubench version 1.1, IPACS. `http://www.ipacs-benchmark.org`, accessed: 2014-09-20.

[11] Fike, J. A., Alonso, J. J., 2011. The development of hyper-dual numbers for exact second-derivative calculations. AIAA 2011-866.

[12] Funaro, D., Gottlieb, D., 1988. A new method of imposing boundary conditions in pseudospectral approximations of hyperbolic equations. Math. Comput. 51, 599–613.

[13] Hicken, J. E., Zingg, D. W., 2008. A parallel Newton-Krylov solver for the Euler equations discretized using simultaneous approximation terms. AIAA J. 46 (11), 2773–2786.

[14] Hicken, J. E., Zingg, D. W., 2010. A simplified and flexible variant of GCROT for solving nonsymmetric linear systems. SIAM J. Sci. Comput. 32 (3), 1672–1694.
[15] Jameson, A., Schmidt, W., Turkel, E., June 1981. Numerical solution of the Euler equations by finite-volume methods using Runge-Kutta time-stepping schemes. AIAA-1981-1259.
[16] Kreiss, H., Scherer, G., 1974. Finite element and finite difference methods for hyperbolic partial differential equations. In: de Boor, C. (Ed.), Mathematical Aspects of Finite Elements in Partial Differential Equations: proceedings of a symposium conducted by the Mathematics Research Center, the University of Wisconsin. Mathematics Research Centre, the University of Wisconsin, Academic Press, pp. 195–212.
[17] Kreyszig, E., 1959. Differential Geometry. University of Toronto Press, Toronto, Ontario, Canada.
[18] Lahmam, H., Cadou, J. M., Zahrouni, H., Damil, N., Potier-Ferry, M., 2002. High-order predictor-corrector algorithms. Int. J. Numer. Meth. Eng. 55, 685–704.
[19] Lee, J., Chiang, H.-D., 2001. Constructive homotopy methods for finding all or multiple DC operating points of nonlinear circuits and systems. IEEE T. Circuits-I 48 (1), 35–50.
[20] Lomax, H., Pulliam, T. H., Zingg, D. W., 2001. Fundamentals of Computational Fluid Dynamics. Springer-Verlag.
[21] Lundberg, B. N., Poore, A. B., 1991. Variable order Adams-Bashforth predictors with an error stepsize control for continuation methods. SIAM J. Sci. Stat. Comp. 12 (3), 695–723.
[22] Lyness, J. N., Moler, C. B., 1967. Numerical differentiation of analytic functions. SIAM J. Numer. Anal. 4 (2), 202–210.
[23] Mackens, W., 1989. Numerical differentiation of implicitly defined space curves. Computing 41, 237–260.
[24] Martins, J. R. R. A., Sturdza, P., Alonso, J. J., 2003. The complex-step derivative approximation. ACM Transactions on Mathematical Software 29 (3), 245–262.
[25] McCroskey, W. J., October 1987. A critical assessment of wind tunnel results for the NACA 0012 airfoil. Tech. rep., Ames Research Center, Moffett Field, California, nASA TM 100019.
[26] Newman III, J. C., Whitfield, D. L., Anderson, W. K., 2003. Step-size independent approach for multidisciplinary sensitivity analysis. J. Aircraft 40 (3), 566–573.
[27] Nielsen, E. J., Anderson, W. K., Walters, R. W., Keyes, D. E., June 1995. Application of Newton-Krylov methodology to a three-dimensional unstructured Euler code. AIAA-95-1733.
[28] Osusky, M., Zingg, D. W., 2013. A parallel Newton-Krylov-Schur flow solver for the Navier-Stokes equations discretized using summation-by-parts operators. AIAA J. 51 (12), 2833–2851.
[29] Pönisch, G., Schwetlick, H., 1981. Computing turning points of curves implicitly defined by nonlinear equations depending on a parameter. Computing 26, 107–121.
[30] Porteous, I. R., 2001. Geometric Differentiation: For the Intelligence of Curves and Surfaces, 2nd Edition. Cambridge University Press.
[31] Riley, D. S., Winters, K. H., 1990. A numerical bifurcation of natural convection in a tilted two-dimensional porous cavity. J. Fluid Mech. 215, 309–329.
[32] Saad, Y., 2003. Iterative Methods for Sparse Linear Systems, 2nd Edition. SIAM, Philadelphia, PA.
[33] Sanchez, J., Marques, F., Lopez, J. M., 2002. A continuation and bifurcation technique for Navier-Stokes flows. J. Comput. Phys. 180, 78–98.
[34] Schwetlick, H., Cleve, J., 1987. Higher order predictors and adaptive steplength control in path following algorithms. SIAM J. Numer. Anal. 24 (6), 1382–1393.
[35] Squire, W., Trapp, G., 1998. Using complex variables to estimate the derivatives of real functions. SIAM Rev. 40, 110–112.
[36] Syam, M. I., Siyyam, H. I., 1999. Numerical differentiation of implicitly defined curves. J. Comput. Appl. Math. 108, 131–144.
[37] Thompson, J. M. T., 1968. The non-linear perturbation analysis of discrete structural systems. Int. J. Solids Struct. 4, 757–768.
[38] Ushida, A., Yamagami, Y., Nishio, Y., Kinouchi, I., Inoue, Y., 2002. An efficient algorithm for finding multiple DC solutions based on the SPICE-oriented Newton homotopy method. IEEE T. Comput. Aid. D. 21 (3), 337–348.
[39] Wales, C., Gaitonde, A. L., Jones, D. P., Avitabile, D., Champneys, A. R., 2012. Numerical continuation of high Reynolds number external flows. International Journal for Numerical Methods in Fluids 68 (2), 135–159.
[40] Winters, K. H., 1987. A bifurcation study of laminar flow in a curved rectangular cross-section. J. Fluid Mech. 180, 343–369.
[41] Winters, K. H., Cliffe, K. A., 1985. The prediction of critical points for thermal explosions in a finite volume. Combust. Flame 62 (1), 13–20.
[42] Zoghbi, A., Stojmenović, I., 1998. Fast algorithms for generating integer partitions. Int. J. Comput. Math. 70, 319–332.

Appendix A. Inversion of a Sparse Matrix with a Dense Row and a Dense Column

A method is derived for solving a linear system of the form

$$\begin{pmatrix} \mathcal{A} & \mathbf{v}_1 \\ \mathbf{v}_2^T & C \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \mathbf{y}_1 \\ y_2 \end{pmatrix}, \tag{A.1}$$

where

$$\mathcal{A} \in \mathbb{R}^{N\times N},\ \mathbf{v}_1, \mathbf{v}_2, \mathbf{y}_1, \mathbf{y}_2 \in \mathbb{R}^N,\ C, x_2, y_2 \in \mathbb{R}.$$

It is assumed that $\mathcal{A}$ is sparse and invertible, whereas $\mathbf{v}_1$ and $\mathbf{v}_2$ are dense, making direct inversion of the augmented matrix expensive.

Expanding the first N rows of the linear system (A.1) gives the equation

$$\mathcal{A}\mathbf{x}_1 + x_2\mathbf{v}_1 = \mathbf{y}_1, \tag{A.2}$$

which can alternatively be written as

$$\mathcal{A}(\mathbf{x}_1 + x_2\tilde{\mathbf{v}}_1) = \mathbf{y}_1, \tag{A.3}$$

where $\tilde{\mathbf{v}}_1 \in \mathbb{R}^N$ is defined implicitly by

$$\mathcal{A}\tilde{\mathbf{v}}_1 = \mathbf{v}_1. \tag{A.4}$$

Let

$$\tilde{\mathbf{x}}_1 = \mathbf{x}_1 + x_2\tilde{\mathbf{v}}_1. \tag{A.5}$$

Then, taking the inner product of both sides with $\tilde{\mathbf{v}}_2$:

$$\mathbf{v}_2 \cdot \tilde{\mathbf{x}}_1 = \mathbf{v}_2 \cdot (\mathbf{x}_1 + x_2\tilde{\mathbf{v}}_1). \tag{A.6}$$

An additional equation is extracted from the $n+1$st row of the linear system (A.1):

$$\tilde{\mathbf{v}}_2 \cdot \mathbf{x}_1 + Cx_2 = \mathbf{y}_2, \tag{A.7}$$

which is used with equation (A.6) to obtain:

$$\mathbf{v}_2 \cdot \tilde{\mathbf{x}} = y_2 - Cx_2 + x_2\mathbf{v}_2 \cdot \tilde{\mathbf{v}}_1. \tag{A.8}$$

This can be solved for x_2:

$$x_2 = \frac{\mathbf{v}_2 \cdot \tilde{\mathbf{x}}_1 - y_2}{\mathbf{v}_2 \cdot \tilde{\mathbf{v}}_1 - C}. \tag{A.9}$$

Substituting this back into equation (A.5):

$$\mathbf{x}_1 = \tilde{\mathbf{x}}_1 - x_2\tilde{\mathbf{v}}_1, \tag{A.10}$$

which completes the derivation. The calculation is summarized in Algorithm 3.

Algorithm 3: An algorithm for the inversion of a sparse matrix with a dense row and a dense column; the linear system is assumed to be of the form of equation (A.1)

Solve the linear system $\mathcal{A}\tilde{\mathbf{v}}_1 = \mathbf{v}_1$ for $\tilde{\mathbf{v}}_1$
Solve the linear system $\mathcal{A}\tilde{\mathbf{x}}_1 = \mathbf{y}_1$ for $\tilde{\mathbf{x}}_1$
$x_2 \leftarrow \frac{\mathbf{v}_2 \cdot \tilde{\mathbf{x}}_1 - y_2}{\mathbf{v}_2 \cdot \tilde{\mathbf{v}}_1 - C}$
$\mathbf{x}_1 \leftarrow \tilde{\mathbf{x}}_1 - x_2\tilde{\mathbf{v}}_1$

Appendix B. Detailed Pseudo-Codes

Algorithm 4: First-order accurate nth directional derivative calculation with direction vectors specific to the order n curve derivative calculation; the variable n_t is the order of the tensor and the integer-valued vector $i_v \in \mathbb{Z}^{n_t}$ contains the orders of derivatives as input; for example, $\nabla^3\mathcal{H}\,(\mathbf{q}, \lambda)\,[\ddot{c}, \dot{c}, \dot{c}]$ is characterized by $n_t = 3$, $i_v = (2, 1, 1)$

Data: n_t, i_v, $\epsilon_1, \ldots, \epsilon_{n-1}$, $\mathbf{q}$, $\dot{\mathbf{q}}, \ldots, \mathbf{q}^{(n-1)}$, λ, $\dot{\lambda}, \ldots, \lambda^{(n)}$
Result: $\mathbf{w}_n$
$\mathbf{w} \leftarrow \mathbf{0}$
for $j = 1 : 2^{n_t}$ **do**
  $n_p \leftarrow 0$
  for $d = 1 : n_t$ **do**
    `/* The following logic ensures that every combination of input vectors is constructed */`
    if $\mod\left(j - 1, 2^d\right) + 1 > 2^{d-1}$ **then**
      $\mathbf{q} \leftarrow \mathbf{q} + (-1)^d\, \epsilon_{i_v(d)} \mathbf{q}^{(i_v(d))}$ and $\lambda \leftarrow \lambda + (-1)^d\, \epsilon_{i_v(d)} \lambda^{(i_v(d))}$
      $n_p \leftarrow n_p + 1$
    end
  end
  Evaluate $\mathcal{H}$ at the perturbed $\mathbf{q}$ and λ
  if $\mod\,(n_t, 4) \leq 1$ **then**
    $\mathbf{w} \leftarrow \mathbf{w} + (-1)^{n_p}\, \mathcal{H}$
  else
    $\mathbf{w} \leftarrow \mathbf{w} + (-1)^{n_p+1}\, \mathcal{H}$
  end
  Reset $\mathbf{q}$ and λ back to their initial values
end
$\mathbf{w} \leftarrow \frac{1}{\epsilon_{v(1)} \cdots \epsilon_{i_v(n_t)}} \mathbf{w}$

Algorithm 5: First-order accurate nth directional derivative calculation with direction vectors specific to the order n curve derivative calculation in the special case where all direction vectors are the same; the variable n_t is the order of the tensor

Data: n_t, i, ϵ_i, $\mathbf{q}$, $\mathbf{q}^{(i)}$, λ, $\lambda^{(i)}$
Result: $\mathbf{w}_n$
$\mathbf{w} \leftarrow \mathbf{0}$
$C_{\mathcal{H},:} \leftarrow 0$
for $j = 1 : 2^{n_t}$ **do**
  $n_p \leftarrow 0$
  $C_\epsilon \leftarrow 0$
  /* Get the coefficients in front of ϵ_i and $\mathcal{H}$, denoted C_ϵ and $C_{\mathcal{H},:}$ respectively, for this term */
  for $d = 1 : n_t$ **do**
    if $\mod\left(j-1, 2^d\right) + 1 > 2^{d-1}$ **then**
      $C_\epsilon \leftarrow C_\epsilon + (-1)^{n_t+d}$
      $n_p \leftarrow n_p + 1$
    end
  end
  $k \leftarrow C_\epsilon + n_t + 1$
  $C_{\mathcal{H},k} \leftarrow C_{\mathcal{H},k} + (-1)^{n_p+1}$
end
for $k = 1 : 2n_t + 1$ **do**
  if $C_{\mathcal{H},k} \neq 0$ **then**
    $C_\epsilon \leftarrow k - n_t - 1$
    $\mathbf{q} \leftarrow \mathbf{q} + C_\epsilon \epsilon_i \mathbf{q}^{(i)}$ and $\lambda \leftarrow \lambda + C_\epsilon \epsilon_i \lambda^{(i)}$
    Evaluate $\mathcal{H}$ at the perturbed $\mathbf{q}$ and λ
    $\mathbf{w} \leftarrow \mathbf{w} + C_{\mathcal{H},k}\mathcal{H}$
    Reset $\mathbf{q}$ and λ back to their initial values
  end
end
$\mathbf{w} \leftarrow \frac{1}{\epsilon_i^{n_t}}\mathbf{w}$

Algorithm 6: Second-order accurate nth directional derivative calculation with direction vectors specific to the order n curve derivative calculation; the variable n_t is the order of the tensor and the integer-valued vector $i_v \in \mathbb{Z}^{n_t}$ contains the orders of derivatives as input; for example, $\nabla^3\mathcal{H}\left(\mathbf{q}, \lambda\right)[\ddot{c}, \dot{c}, \dot{c}]$ is characterized by $n_t = 3$, $i_v = (2, 1, 1)$

Data: n_t, i_v, $\epsilon_1, \ldots, \epsilon_{n-1}$, $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ldots$, $\mathbf{q}^{(n-1)}$, λ, $\dot{\lambda}$, $\ldots$, $\lambda^{(n)}$
Result: $\mathbf{w}_n$
$\mathbf{w} \leftarrow \mathbf{0}$
for $j = 1 : 2^{n_t}$ **do**
  $C_i \leftarrow 1$
  for $d = 1 : n_t$ **do**
    `/* The following logic ensures that every combination of input vectors is constructed */`
    if $\mod\left(j - 1, 2^d\right) + 1 > 2^{d-1}$ **then**
      $C_\epsilon \leftarrow -1$
      $C_i \leftarrow -C_i$
    else
      $C_\epsilon \leftarrow 1$
    end
    $\mathbf{q} \leftarrow \mathbf{q} + C_\epsilon \epsilon_{i_v(d)} \mathbf{q}^{(i_v(d))}$ and $\lambda \leftarrow \lambda + C_\epsilon \epsilon_{i_v(d)} \lambda^{(i_v(d))}$
  end
  Evaluate $\mathcal{H}$ at the perturbed $\mathbf{q}$ and λ
  $\mathbf{w} \leftarrow \mathbf{w} + C_i \mathcal{H}$
  Reset $\mathbf{q}$ and λ back to their initial values
end
$\mathbf{w} \leftarrow \frac{1}{2^{n_t}\left(\epsilon_{i_v(1)} \cdots \epsilon_{i_v(n_t)}\right)} \mathbf{w}$

Algorithm 7: Second-order accurate nth directional derivative calculation with direction vectors specific to the order n curve derivative calculation in the special case where all direction vectors are the same; the variable n_t is the order of the tensor

Data: n_t, i, ϵ_i, $\mathbf{q}$, $\mathbf{q}^{(i)}$, λ, $\lambda^{(i)}$
Result: $\mathbf{w}_n$
$\mathbf{w} \leftarrow \mathbf{0}$
$C_{\mathcal{H},:} \leftarrow 0$
for $j = 1 : 2^{n_t}$ **do**
  $C_\epsilon \leftarrow 0$
  $C_i \leftarrow 1$
  `/* Get the coefficients in front of` ϵ_i `and` $\mathcal{H}$`, denoted` C_ϵ `and` $C_{\mathcal{H},:}$ `respectively, for this term */`
  for $d = 1 : n_t$ **do**
    if $\mod\left(j-1, 2^d\right) + 1 > 2^{d-1}$ **then**
      $C_\epsilon \leftarrow C_\epsilon - 1$
      $C_i \leftarrow -C_i$
    else
      $C_\epsilon \leftarrow C_\epsilon + 1$
    end
  end
  $k \leftarrow C_\epsilon + n_t + 1$
  $C_{\mathcal{H},k} \leftarrow C_{\mathcal{H},k} + C_i$
end
for $k = 1 : 2n_t + 1$ **do**
  if $C_{\mathcal{H},k} \neq 0$ **then**
    $C_\epsilon \leftarrow k - n_t - 1$
    $\mathbf{q} \leftarrow \mathbf{q} + C_\epsilon \epsilon_i \mathbf{q}^{(i)}$ and $\lambda \leftarrow \lambda + C_\epsilon \epsilon_i \lambda^{(i)}$
    Evaluate $\mathcal{H}$ at the perturbed $\mathbf{q}$ and λ
    $\mathbf{w} \leftarrow \mathbf{w} + C_{\mathcal{H},k}\mathcal{H}$
    Reset $\mathbf{q}$ and λ back to their initial values
  end
end
$\mathbf{w} \leftarrow \frac{1}{2^{n_t}\epsilon_i^{n_t}}\mathbf{w}$

Algorithm 8: An algorithm for calculating Δs for given λ and $\Delta\lambda$

Data: $\lambda, \dot{\lambda}, \ldots, \lambda^{(n)}, \Delta\lambda, n, f_{\Delta s}, i_{\max}$
Result: Δs
`/* Initialize */`
$\lambda_0 \leftarrow \lambda, \quad \lambda_t \leftarrow \lambda + \Delta\lambda, \quad d \leftarrow n, \quad success \leftarrow 1$
$\Delta s \leftarrow$ equation (65)
`/* Attempt to solve for` Δs`; reduce order if unsuccessful */`
while $d > 1$ && $success = 1$ **do**
  $\lambda_p \leftarrow$ equation (64)
  $success \leftarrow 0$
  `/* Find bisection interval` $[\Delta s_L, \Delta s_U]$ `*/`
  $i \leftarrow 0$
  $\Delta s_L \leftarrow \Delta s, \Delta s_U \leftarrow \Delta s$
  if $\lambda_p > \lambda_t$ **then**
    `/* Get the upper endpoint` Δs_U `*/`
    while $\lambda > \lambda_t$ && $i < i_{\max}$ **do**
      $i \leftarrow i + 1$
      $\lambda \leftarrow$ equation (64)
      $\Delta s_U \leftarrow \Delta s_U + f_{\Delta s}\Delta s_U$
      if $i = i_{\max}$ || $\lambda > \lambda_p$ **then**
        $success \leftarrow 0$
      end
    end
  else if $\lambda_p < \lambda_t$ **then**
    `/* Get the lower endpoint` Δs_L `*/`
    while $\lambda < \lambda_t$ && $i < i_{\max}$ **do**
      $i \leftarrow i + 1$
      $\lambda \leftarrow$ equation (64)
      $\Delta s_L \leftarrow \Delta s_L - f_{\Delta s}\Delta s_L$
      if $i = i_{\max}$ || $\lambda < \lambda_p$ **then**
        $success \leftarrow 0$
      end
    end
  end
  if $success = 0$ **then**
    `/* Reduce the degree by one and cycle the loop */`
    $d \leftarrow d - 1$
  end
end
Solve for Δs using the bisection method on the interval $[\Delta s_L, \Delta s_U]$