THE DISCRETE ADJOINT METHOD FOR LARGE-SCALE OPTIMIZATION PROBLEMS WITH LINEAR TIME-DEPENDENT PDE CONSTRAINTS

KAI ROTHAUGE*, ELDAD HABER†, AND URI ASCHER‡

Abstract. We discuss the application of the discrete adjoint method to large-scale distributed parameter-estimation and sensitivity analysis problems that are constrained by linear time-dependent PDEs. The PDEs are solved using an arbitrary-order linear multistep or Runge-Kutta method, making the resulting procedures for the computation of the action of the sensitivity matrix and of the Hessian well-suited to situations where the forward problem needs to be solved to high accuracy. These procedures will depend on the time-stepping method being used and we discuss their implementation in detail. Adjoint linear multistep methods are seen to be equivalent to the corresponding forward schemes apart from the handling of the source terms, and for adjoint Runge-Kutta methods we give a simple argument to show that their order of accuracy is the same as that of the corresponding forward methods. This property is illustrated with a numerical example. The approach used here can easily be adapted to nonlinear PDEs solved using other types of time-stepping methods.

Key words. Parameter estimation, model calibration, sensitivity analysis, inverse problems, adjoint method, time-dependent PDE, gradient-based optimization, linear multistep methods, Runge-Kutta methods

AMS subject classifications. Partial Differential Equations, Numerical Analysis, Optimization

1. Introduction.

The problem of recovering parameter functions that appear in partial differential equations (PDEs) from a set of measurements is an important type of inverse problem; see e.g. [4, 36, 5], and see e.g. [22] for a more general discussion of inverse problems for PDEs. The parameter estimation problem has applications in many branches of science and engineering, for instance geophysics [11, 16], fluid dynamics [23], computer vision [35] and bioscience [8], to name just a few.

We are interested in recovering the discretized set of parameters $\mathbf{p} = \begin{bmatrix}\mathbf{m}^\top & \mathbf{s}^\top\end{bmatrix}^\top$, with $\mathbf{m} \in \mathbb{R}^{N_\mathbf{m}}$ representing model parameters and $\mathbf{s} \in \mathbb{R}^{N_\mathbf{s}}$ source parameters. The standard approach is to formulate the parameter estimation problem as a PDE-constrained optimization problem of the form

$$\begin{aligned} \underset{\mathbf{p}}{\arg\min} \quad & \mathcal{M}(\mathbf{d}(\mathbf{y}(\mathbf{p})), \mathbf{d}^{\text{obs}}) + \beta\, \mathcal{R}(\mathbf{p}) \\ \text{subject to} \quad & \mathbf{T}(\mathbf{y}(\mathbf{p}), \mathbf{m}) = \mathbf{q}(\mathbf{s}), \end{aligned} \tag{1.1}$$

where $\mathcal{R}$ is a differentiable regularization function, β a regularization parameter and $\mathcal{M}$ is a differentiable misfit function that in some way quantifies the difference between the actual observations $\mathbf{d}^{\text{obs}}$ and the simulated observations $\mathbf{d}$, where we assume the number of observations to be large. The simulated measurements are obtained from the *forward solution* $\mathbf{y}$, which is the solution we get after solving the fully discretized PDE constraint, represented by $\mathbf{T}(\mathbf{m}) = \mathbf{q}(\mathbf{s})$.

A closely related problem is that of sensitivity analysis, where the dependence of the simulated data $\mathbf{d}$ on the parameters is investigated. Both of these applications require the action[1] of the *sensitivity matrix*

$$\mathbf{J} := \frac{\partial \mathbf{d}}{\partial \mathbf{p}} = \begin{bmatrix} \dfrac{\partial \mathbf{d}}{\partial \mathbf{m}} & \dfrac{\partial \mathbf{d}}{\partial \mathbf{s}} \end{bmatrix} \tag{1.2}$$

*Department of Mathematics, University of British Columbia (rothauge@math.ubc.ca)
†Department of Earth and Ocean Science, University of British Columbia (haber@eos.ubc.ca)
‡Department of Computer Science, University of British Columbia (ascher@cs.ubc.ca)

[1] By the action of a matrix we mean the product of the matrix with an arbitrary appropriately-sized vector.

or its transpose, and for large-scale applications, where both the number of observations and the number of parameters are large, it is paramount that this computation is done as efficiently as possible.

In this paper we restrict ourselves to linear PDEs due to space constraints, but the approach used here can easily be applied to nonlinear PDEs. A common technique for numerically solving PDEs is known as the method of lines, where we semi-discretize the PDE using some spatial discretization scheme and incorporate appropriate and known boundary conditions. This leads to a large system of ODEs that can be written in generic first-order form as

$$\begin{aligned} \frac{\partial \widehat{\mathbf{y}}(t)}{\partial t} &= \mathbf{L}(\mathbf{m})\,\widehat{\mathbf{y}}(t) + \widehat{\mathbf{q}}(t;\mathbf{s}), \qquad 0 \leq t \leq T \\ \widehat{\mathbf{y}}(0) &= \mathbf{y}_0, \end{aligned} \tag{1.3}$$

where $\widehat{\mathbf{y}}$ and $\widehat{\mathbf{q}}$ are discrete in space, with N unknowns, but still continuous in time. It is assumed that the coefficients appearing in the PDE are stationary, which is not a significant restriction since many problems of interest involve material parameter functions that vary in space but not in time. The number of parameters $N_{\mathbf{p}} = N_{\mathbf{m}} + N_{\mathbf{s}}$ generally depends on the spatial resolution of the underlying grid.

To solve (1.3), we employ a potentially high-order time-stepping method and the resulting temporal discretization allows us to represent (1.3) by

$$\mathbf{T}(\mathbf{y},\mathbf{m}) = \widehat{\mathbf{T}}(\mathbf{m})\,\mathbf{y} + \mathbf{S}(\mathbf{m})\,\mathbf{y}_0 = \mathbf{q}(\mathbf{s}), \tag{1.4}$$

where the structures of $\widehat{\mathbf{T}}$, $\mathbf{S}$ and $\mathbf{q}(\mathbf{s})$ depend on the particular time-stepping method. The two most popular classes of time-stepping schemes are linear multistep (LM) and Runge-Kutta (RK) methods (e.g. [12, 18, 19, 3]) and we consider both of these in this article.

We will not restrict our discussion to any particular misfit function and we only require that it is differentiable with respect to $\mathbf{d}$. There are many to choose from, the most popular being the least squares function $\mathcal{M} = \frac{1}{2}\left\| \mathbf{d} - \mathbf{d}^{\text{obs}} \right\|^2$. A common example of a regularization function is Tikhonov-type regularization [37, 10]. The derivatives of $\mathcal{R}$ with respect to $\mathbf{p}$ are generally well-known and we do not address them here.

Since $\mathcal{M}$ is assumed to be differentiable, we can apply gradient-based optimization methods ([27, 9]) to minimize (1.1), which are iterative methods that require the gradient of $\mathcal{M}$ with respect to $\mathbf{p}$ to find a local minimum, and for some methods one might even require the action of the Hessian of $\mathcal{M}$, or (more likely in practice) the action of an approximation of the Hessian.

To calculate the sensitivity matrix and the derivatives of $\mathcal{M}$, we turn to the *adjoint state method*, which is an important and efficient technique used for this purpose in many applications, see for instance [30] for a review of the method applied to geophysical problems, [15, 26] for some uses in CFD and [7] for some other examples of applications.

As we will discuss, the adjoint method requires us to find derivatives of $\mathbf{T}(\mathbf{y},\mathbf{m})$ with respect to $\mathbf{y}$ and $\mathbf{m}$. We will also need to solve the *adjoint problem* $\widehat{\mathbf{T}}^{\top}\boldsymbol{\lambda} = \boldsymbol{\theta}$, where $\boldsymbol{\lambda}$ and $\boldsymbol{\theta}$ are known as the adjoint solution and the adjoint source, respectively. This is done by using an *adjoint time-stepping method* that is solved backward in time and that corresponds to the forward time-stepping method used to solve (1.3). Clearly both the derivatives of $\mathbf{T}(\mathbf{y},\mathbf{m})$ and the adjoint time-stepping method lead to expressions that are highly dependent on the structure of the time-stepping system (1.4).

Our goal is to systematically derive these expressions for general LM and RK schemes so that this article can serve as a reference guide for possible readers who wish to apply the

adjoint method to problems that require high-order time-stepping methods, although implementations using low-order schemes can also benefit. So that a variety of practitioners may benefit, we try to keep the discussion accessible and do not assume much background.

It is important to point out that the adjoint method is often applied directly to the semi-discrete system (1.3), or sometimes even to a completely continuous formulation, leading to a semi-discrete (or continuous) adjoint problem (see for instance [11, 29]) that is then solved using some time-stepping method which may differ from the one used for solving (1.3), thus decoupling the discretization process of the forward problem from that of its adjoint. Expressions for the derivatives of $\mathcal{M}$ are also derived in a semi-discrete or continuous setting and then subsequently discretized. This falls within the framework of optimize-then-discretize (OD), which may be beneficial in some scenarios since it does not require the particular time-stepping method used by the forward problem to be taken into account when solving the adjoint problem. While the resulting comparative ease of implementation is useful, this approach can lead to significant differences in the computed gradients (as illustrated in [15], see also [25]), and the resulting inaccuracy does affect the minimization algorithm when this happens, especially on coarse grids.

Our view is that the discretize-then-optimize (DO) framework used here is the preferable approach to the adjoint problem in most cases. This has the drawback that one needs to specify the adjoint problem for the fully discretized (1.4), and, since the forward solution in this case depends directly on the time-stepping scheme being used, the adjoint method will also depend on the time-stepping scheme, which leads to the implementation difficulties we address in this paper.

An alternate approach to computing the gradient that falls inside the DO framework is automatic differentiation (AD), where the exact derivatives (up to floating-point error) of $\mathcal{M}$ with respect to $\mathbf{p}$ are computed automatically by following the sequence of arithmetic commands used in the computation of $\mathcal{M}$ and successively applying basic differentiation rules, particularly the chain rule, to determine the dependence of $\mathcal{M}$ on $\mathbf{m}$.

This approach is a great tool in numerous applications. However, it has some drawbacks, for instance AD in reverse mode tends to use a significant amount of memory that needs to be carefully managed, and this can render large scale applications not feasible. This limitation commonly arises in our applications where 4D problems are common. A second problem is of efficiency, since AD codes are "black box" codes and do not use information about the discrete problem that can be used in order to obtain very efficient code. Finally, computing the action of large sensitivity matrices is also known to be particularly challenging for AD. Industrial codes that require efficiency both in memory and computational time can therefore do better when computing the derivatives using the adjoint method.

For these reasons we do not consider AD any further here, but see [14] for a discussion on AD vs. the continuous adjoint method, and see [31] for a joint adjoint-AD implementation to compute Hessian matrices. [38] addresses AD for explicit RK schemes in the context of optimal control.

Applying the discrete adjoint method to specific, and usually lower-order, time-stepping methods has of course been done before (see for instance [1, 24, 20] for applications to the Crank-Nicolson method), but our discussion is not limited to any particular order of the time-stepping method and therefore has wide applicability when the PDEs are linear. Sanz-Serna has recently [33] discussed the adjoint method in conjunction with symplectic Runge-Kutta methods in the context of optimal control.

The rest of the article is structured as follows. In Section 2 we introduce the necessary notation and show how (1.3) can be written in the form of (1.4) for LM and RK methods. Section 3 reviews the adjoint method and illustrates how it is used in the computation the

action of sensitivity matrix and its transpose, with an application to the computation of the gradient of $\mathcal{M}$. We also include a brief discussion on the computation of the action of the Hessian, and a derivation of the required expression for this is given in the appendix. In Section 4 we concern ourselves with the adjoint LM and RK time-stepping methods, and we also talk about the convergence properties of the adjoint RK schemes. The order of accuracy for adjoint RK methods has already been discussed in [17] for nonlinear problems in the context of optimal control, but we give a much simpler explanation as to why the adjoint RK schemes for linear problems must have the same order of accuracy as the corresponding forward methods. In Section 5 we find the relevant derivatives of $\mathbf{T}$ and in Section 6 we perform a simple numerical experiment using the acoustic wave equation. Some final remarks and avenues for future work are given in Section 7.

2. Time-Stepping Methods. In this section we show how to represent LM and RK methods by large linear systems in the form of (1.4), which is based on the approach used in [2]. The matrices $\widehat{\mathbf{T}}$ and $\mathbf{S}$ are purely conceptual and are never formed in practice, but as we will see in Section 3, they are a powerful tool when used in conjunction with the adjoint method. It is important in later sections to be aware of the fact that $\mathbf{y}$ depends on $\mathbf{m}$ and $\mathbf{s}$ indirectly through $\widehat{\mathbf{T}}$, $\mathbf{S}$ and $\mathbf{q}$, although for notational simplicity we will suppress this dependence in this section. The initial condition $\mathbf{y}_0$ is assumed to be known and independent of $\mathbf{m}$ and $\mathbf{s}$.

The time-stepping schemes we consider here include implicit methods, which require the solution of a linear system at each time step. For large-scale problems the cost of solving a linear system might be prohibitive and therefore these methods are rarely used, but we include them here to keep the discussion general.

2.1. Multistep Methods. Given a linear ODE system in the form (1.3), the general linear s-step method [3, 18, 19] at time step k is written as

$$\sum_{j=0}^{s} \alpha_j^{(s)}\, \widehat{\mathbf{y}}(t_{k+1-j}) = \tau \sum_{j=0}^{s} \beta_j^{(s)}\, \mathbf{L}\, \widehat{\mathbf{y}}(t_{k+1-j}) + \tau \sum_{j=0}^{s} \beta_j^{(s)}\, \widehat{\mathbf{q}}(t_{k+1-j}) \tag{2.1}$$

where $s \leq k < K$ and τ is a uniform step size. Here $\alpha_j^{(s)}$ and $\beta_j^{(s)}$ are weights that determine the s-step method, with $\alpha_0^{(s)} = 1$ (always). There are two main classes of multistep methods in active use: *backward differentiation formulas* (BDFs) for which $\beta_1^{(s)} = \cdots = \beta_s^{(s)} = 0$, and *Adams*-type methods for which $\alpha_1 = -1$ and $\alpha_2 = \cdots = \alpha_s = 0$. Further, *Adams-Bashforth* methods are explicit ($\beta_0^{(s)} = 0$) and *Adams-Moulton* methods are implicit ($\beta_0^{(s)} \neq 0$).

An s-step method needs to have the solution at s previous time steps available, which is not the case at the start of the integration, i.e., when $k < s$. There are various options to handle this, of which we mention one: use lower-order LM methods to gradually build up a solution for the first $s-1$ time steps. One can use methods with increasingly higher orders of accuracy, for instance employ a first-order method to integrate up to τ, then a second-order method to integrate up to 2τ (using the previously computed solution at τ), etc. If necessary, the lower-order schemes can have time steps that are smaller than τ, thereby ensuring that the solution at $k\tau$ is sufficiently accurate.

In order to represent the time-stepping scheme in the form of (1.4), we introduce the $KN \times KN$ block template matrix

$$\times = \times^{(s)} \otimes \mathbf{I}_N,$$

where $\times^{(s)}$ is the placeholder of a $K \times K$ matrix of the form

$$\times^{(s)} = \begin{bmatrix} \boxed{\times^{(1:s)}} & & & & & & & \\ \times_{s-1} & \cdots & \times_1 & \times_0 & & & & \\ \times_s & \times_{s-1} & \cdots & \times_1 & \times_0 & & & \\ & \ddots & \ddots & & \ddots & \ddots & & \\ & & \times_s & \times_{s-1} & \cdots & \times_1 & \times_0 & \\ & & & \ddots & \ddots & & \ddots & \ddots \\ & & & & \times_s & \times_{s-1} & \cdots & \times_1 & \times_0 \end{bmatrix}.$$

Using this template, $\mathbf{A}^{(s)}$, $\mathbf{A}^{(1:s)}$ and $\mathbf{A}$ are simply $\times^{(s)}$, $\times^{(1:s)}$ and $\times$ when we set $\times_j = \alpha_j^{(s)}$, and likewise setting $\times_j = \beta_j^{(s)}$ gives $\mathbf{B}^{(s)}$, $\mathbf{B}^{(1:s)}$ and $\mathbf{B}$.

The *transition blocks* $\mathbf{A}^{(1:s)}$ and $\mathbf{B}^{(1:s)}$ in $\mathbf{A}^s$ and $\mathbf{B}^s$ are introduced to allow for the approach of handling the initialization mentioned above.

For simplicity assume that each of the lower-order methods uses the same time step τ as the higher-order method. In this case the transition blocks are

$$\times^{(1:s)} = \begin{bmatrix} \times_0^{(1)} & & & \\ \times_1^{(2)} & \times_0^{(2)} & & \\ \vdots & \ddots & \ddots & \\ \times_{s-2}^{(s-1)} & \cdots & \times_1^{(s-1)} & \times_0^{(s-1)} \end{bmatrix},$$

with the superscripts (σ) indicating the number of steps required by the method used at that time step (for $0 < k < s$), with $\alpha_j^{(\sigma)}$ and $\beta_j^{(\sigma)}$ being the corresponding weights of the σ-step method.

Getting back to a compact representation of the time-stepping system, we also define the $KN \times N$ block matrices

$$\boldsymbol{\alpha}^\top = \begin{bmatrix} \alpha_1^{(1)} & \alpha_2^{(2)} & \dots & \alpha_s^{(s)} & \mathbf{0}_{1\times(K-s)} \end{bmatrix} \otimes \mathbf{I}_N$$
$$\boldsymbol{\beta}^\top = \begin{bmatrix} \beta_1^{(1)} & \beta_2^{(2)} & \dots & \beta_s^{(s)} & \mathbf{0}_{1\times(K-s)} \end{bmatrix} \otimes \mathbf{I}_N.$$

Then (2.1) can be written as

$$\mathbf{A}\,\mathbf{y} = \tau\,\mathbf{B}\,\mathrm{diag}(\mathbf{L})\,\mathbf{y} + \tau\,\boldsymbol{\beta}\,\mathbf{L}\,\mathbf{y}_0 - \boldsymbol{\alpha}\,\mathbf{y}_0 + \mathbf{q}, \tag{2.2}$$

with $\mathrm{diag}(\mathbf{L}) = \mathbf{I}_K \otimes \mathbf{L}$, $\mathbf{y} = \begin{bmatrix} \mathbf{y}_1^\top & \dots & \mathbf{y}_K^\top \end{bmatrix}^\top$, where $\mathbf{y}_k = \widehat{\mathbf{y}}(t_k)$, and $\mathbf{q} = \begin{bmatrix} \mathbf{q}_1^\top & \dots & \mathbf{q}_K^\top \end{bmatrix}^\top$, where $\mathbf{q}_k = \tau \sum_{j=0}^{s} \beta_j^{\min(k,s)}\, \widehat{\mathbf{q}}(t_{k+1-j})$.

Setting $\widehat{\mathbf{T}} = \mathbf{A} - \tau\,\mathbf{B}\,\mathrm{diag}(\mathbf{L})$ and $\mathbf{S} = \boldsymbol{\alpha} - \tau\,\boldsymbol{\beta}\,\mathbf{L}$, (2.2) can be written as (1.4). The transition blocks $\mathbf{A}^{(1:s)}$ and $\mathbf{B}^{(1:s)}$, solution $\mathbf{y}$ and source term $\mathbf{q}$ can be modified in an obvious way if lower-order methods at the start of the integration use time steps that are smaller than τ. We will not consider this more general formulation here due to space constraints.

Examples

The Adams-Bashforth methods with $s = 1, 2, 3$ are

$$
\begin{aligned}
s = 1: \quad & \mathbf{y}_{k+1} = \mathbf{y}_k + \tau\,\mathbf{L}\,\mathbf{y}_k + \mathbf{q}_{k+1}, \qquad \text{where } \mathbf{q}_{k+1} = \tau\,\widehat{\mathbf{q}}(t_k), \\
s = 2: \quad & \mathbf{y}_{k+2} = \mathbf{y}_{k+1} + \tau\,\mathbf{L}\left(\frac{3}{2}\mathbf{y}_{k+1} - \frac{1}{2}\mathbf{y}_k\right) + \mathbf{q}_{k+2} \\
& \text{where } \mathbf{q}_{k+2} = \tau\left(\frac{3}{2}\widehat{\mathbf{q}}(t_{k+1}) - \frac{1}{2}\widehat{\mathbf{q}}(t_k)\right), \\
s = 3: \quad & \mathbf{y}_{k+3} = \mathbf{y}_{k+2} + \tau\,\mathbf{L}\left(\frac{23}{12}\mathbf{y}_{k+2} - \frac{4}{3}\mathbf{y}_{k+1} + \frac{5}{12}\mathbf{y}_k\right) + \mathbf{q}_{k+3} \\
& \text{where } \mathbf{q}_{k+3} = \tau\left(\frac{23}{12}\,\widehat{\mathbf{q}}(t_{k+2}) - \frac{4}{3}\,\widehat{\mathbf{q}}(t_{k+1}) + \frac{5}{12}\,\widehat{\mathbf{q}}(t_k)\right),
\end{aligned}
$$

so if using a three-step Adams-Bashforth method we have

$$
\boldsymbol{\alpha}^\top = \begin{bmatrix} -1 & \mathbf{0}_{1\times(K-1)} \end{bmatrix} \otimes \mathbf{I}_N \qquad \boldsymbol{\beta}^\top = \begin{bmatrix} 1 & -\frac{1}{2} & \frac{5}{12} & \mathbf{0}_{1\times(K-3)} \end{bmatrix} \otimes \mathbf{I}_N
$$

$$
\mathbf{A} = \begin{bmatrix} 1 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & & \\ & & & -1 & 1 \end{bmatrix} \otimes \mathbf{I}_N, \qquad \mathbf{B} = \begin{bmatrix} 0 & & & & & \\ \frac{3}{2} & 0 & & & & \\ -\frac{4}{3} & \frac{23}{12} & 0 & & & \\ \frac{5}{12} & -\frac{4}{3} & \frac{23}{12} & 0 & & \\ & & \ddots & & & \\ & & \frac{5}{12} & -\frac{4}{3} & \frac{23}{12} & 0 \end{bmatrix} \otimes \mathbf{I}_N.
$$

The backward differentiation formulas of orders $s = 1, 2, 3$ are

$$
\begin{aligned}
s = 1: \quad & \mathbf{y}_{k+1} - \mathbf{y}_k = \tau\,\mathbf{L}\,\mathbf{y}_{k+1} + \mathbf{q}_{k+1} \\
& \text{where } \mathbf{q}_{k+1} = \tau\,\widehat{\mathbf{q}}(t_{k+1}), \\
s = 2: \quad & \mathbf{y}_{k+2} - \frac{4}{3}\mathbf{y}_{k+1} + \frac{1}{3}\mathbf{y}_k = \tau\frac{2}{3}\,\mathbf{L}\,\mathbf{y}_{k+2} + \mathbf{q}_{k+2} \\
& \text{where } \mathbf{q}_{k+2} = \tau\frac{2}{3}\,\widehat{\mathbf{q}}(t_{k+2}), \\
s = 3: \quad & \mathbf{y}_{k+3} - \frac{18}{11}\mathbf{y}_{k+2} + \frac{9}{11}\mathbf{y}_{k+1} - \frac{2}{11}\mathbf{y}_k = \tau\frac{6}{11}\,\mathbf{L}\,\mathbf{y}_{k+3} + \mathbf{q}_{k+3} \\
& \text{where } \mathbf{q}_{k+3} = \tau\frac{6}{11}\,\widehat{\mathbf{q}}(t_{k+3}),
\end{aligned}
$$

and therefore the three-step BDF can be represented in matrix form using

$$
\boldsymbol{\alpha}^\top = \begin{bmatrix} -1 & \frac{1}{3} & -\frac{2}{11} & \mathbf{0}_{1\times(K-3)} \end{bmatrix} \otimes \mathbf{I}_N, \qquad \boldsymbol{\beta}^\top = \mathbf{0}_{1\times K} \otimes \mathbf{I}_N
$$

$$
\mathbf{A} = \begin{bmatrix} 1 & & & & \\ -\frac{4}{3} & 1 & & & \\ \frac{9}{11} & -\frac{18}{11} & 1 & & \\ -\frac{2}{11} & \frac{9}{11} & -\frac{18}{11} & 1 & \\ & & \ddots & & \\ & -\frac{2}{11} & \frac{9}{11} & -\frac{18}{11} & 1 \end{bmatrix} \otimes \mathbf{I}_N, \qquad \mathbf{B} = \begin{bmatrix} 1 & & & & \\ & \frac{2}{3} & & & \\ & & \frac{6}{11} & & \\ & & & \ddots & \\ & & & & \frac{6}{11} \end{bmatrix} \otimes \mathbf{I}_N.
$$

2.2. Runge-Kutta Methods. The family of s-stage Runge-Kutta methods for linear problems is given by

$$\mathbf{y}_k = \mathbf{y}_{k-1} + \tau_k \sum_{\sigma=1}^{s} b_\sigma \, \mathbf{Y}_{k,\sigma}, \tag{2.3a}$$

where $\mathbf{y}_k = y(t_k)$ and the internal stages are, for the ODE system (1.3) and $\sigma = 1, \ldots, s$,

$$\mathbf{Y}_{k,\sigma} = \mathbf{L}\,\mathbf{y}_{k-1} + \tau_k \sum_{i=1}^{s} a_{\sigma i}\, \mathbf{L}\,\mathbf{Y}_{k,i} + \widehat{\mathbf{q}}(t_{k-1} + c_\sigma \tau_k). \tag{2.3b}$$

The procedure starts from a known initial condition $\mathbf{y}_0$. Here a particular method is determined by setting the number of stages and the corresponding coefficients $a_{\sigma i}$, weights b_σ and nodes c_σ, which are commonly summarized in a *Butcher tableau*:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array} \quad = \quad \begin{array}{c|c} \mathbf{c}_s & \mathbf{A}_s \\ \hline & \mathbf{b}_s^\top \end{array}.$$

The method is said to be *explicit* if $a_{\sigma i} = 0$ for $\sigma \le i$ and *diagonally-implicit* if $a_{\sigma i} = 0$ for $\sigma < i$. It is *implicit* otherwise. Note that we allow for variable time steps τ_k here, although we will not be able to actually address embedded Runge-Kutta methods with adaptive time steps in this paper due to space constraints.

(2.3a) and(2.3b) can be written in matrix form as

$$\begin{bmatrix} \mathbf{I}_N & & \\ \mathbf{D} & \mathbf{A}_k & \\ -\mathbf{I}_N & \mathbf{B}_k^\top & \mathbf{I}_N \end{bmatrix} \begin{bmatrix} \mathbf{y}_{k-1} \\ \mathbf{Y}_k \\ \mathbf{y}_k \end{bmatrix} = \begin{bmatrix} \mathbf{y}_{k-1} \\ \mathbf{q}_k \\ \mathbf{0}_{N\times 1} \end{bmatrix}, \tag{2.4}$$

with

$$\begin{gathered} \mathbf{D} = -\mathbf{1}_s \otimes \mathbf{L}, \qquad \mathbf{A}_k = \mathbf{I}_{sN} - \tau_k \mathbf{A}_s \otimes \mathbf{L}, \qquad \mathbf{B}_k = -\tau_k \mathbf{b}_s \otimes \mathbf{I}_N \\ \mathbf{Y}_k = \begin{bmatrix} \mathbf{Y}_{k,1} \\ \vdots \\ \mathbf{Y}_{k,s} \end{bmatrix} \qquad \text{and} \qquad \mathbf{q}_k = \begin{bmatrix} \widehat{\mathbf{q}}(t_{k-1} + c_1\tau_k) \\ \vdots \\ \widehat{\mathbf{q}}(t_{k-1} + c_s\tau_k) \end{bmatrix}. \end{gathered} \tag{2.5}$$

Here $\mathbf{1}_s$ is a vector of 1's of length s. The above system constitutes a single time step in the solution procedure, so the RK procedure as a whole can be represented by (1.4) with

$$\widehat{\mathbf{T}} = \begin{bmatrix} \mathbf{A}_1 & & & & & & \\ \mathbf{B}_1^\top & \mathbf{I}_N & & & & & \\ & \mathbf{D} & \mathbf{A}_2 & & & & \\ & -\mathbf{I}_N & \mathbf{B}_2^\top & \mathbf{I}_N & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & \mathbf{D} & \mathbf{A}_K & \\ & & & & -\mathbf{I}_N & \mathbf{B}_K^\top & \mathbf{I}_N \end{bmatrix}, \qquad \mathbf{S} = \begin{bmatrix} \mathbf{D} \\ -\mathbf{I}_N \\ \mathbf{0}_{sN\times N} \\ \mathbf{0}_{N\times N} \\ \vdots \\ \mathbf{0}_{sN\times N} \\ \mathbf{0}_{N\times N} \end{bmatrix}, \tag{2.6}$$

$$\mathbf{y} = \begin{bmatrix} \mathbf{Y}_1^\top & \mathbf{y}_1^\top & \cdots & \mathbf{Y}_K^\top & \mathbf{y}_K^\top \end{bmatrix}^\top$$

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_1^\top & \mathbf{0}_{1\times N} & \cdots & \mathbf{q}_K^\top & \mathbf{0}_{1\times N} \end{bmatrix}^\top.$$

Here $\mathbf{y}, \mathbf{q} \in \mathbb{R}^{(s+1)KN}$, $\mathbf{S} \in \mathbb{R}^{(s+1)KN \times N}$ and $\widehat{\mathbf{T}} \in \mathbb{R}^{(s+1)KN \times (s+1)KN}$.

Notice that we have included to internal stages (2.3b) in $\mathbf{y}$. We do this because they are required when computing derivatives, specifically when we compute the derivatives of $\widehat{\mathbf{T}}\,\mathbf{y}$ with respect to the model parameters, which leads to a significant increase in storage overhead for high-order Runge-Kutta schemes. This can be mitigated by the use of checkpointing, where the solution is stored only for some time steps and the solution at other time steps is then recomputed using these stored solutions as needed, although note that this effectively means that the forward solution has to be calculated twice for every adjoint computation.

Examples

The popular fourth-order Runge-Kutta (RK4) method is given by

$$\mathbf{A}_s = \begin{bmatrix} 0 & & & \\ \frac{1}{2} & 0 & & \\ & \frac{1}{2} & 0 & \\ & & 1 & 0 \end{bmatrix} \qquad \mathbf{b}_s = \begin{bmatrix} \frac{1}{6} \\ \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{6} \end{bmatrix} \quad \text{and} \quad \mathbf{c}_s = \begin{bmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \\ 1 \end{bmatrix},$$

so that at some time step k we have

$$\mathbf{A}_k = \begin{bmatrix} \mathbf{I}_N & & & \\ -\frac{\tau_k}{2}\mathbf{L} & \mathbf{I}_N & & \\ & -\frac{\tau_k}{2}\mathbf{L} & \mathbf{I}_N & \\ & & -\tau_k\mathbf{L} & \mathbf{I}_N \end{bmatrix} \quad \text{and} \quad \mathbf{B}_k = -\begin{bmatrix} \frac{\tau_k}{6}\mathbf{I}_N \\ \frac{\tau_k}{3}\mathbf{I}_N \\ \frac{\tau_k}{3}\mathbf{I}_N \\ \frac{\tau_k}{6}\mathbf{I}_N \end{bmatrix}.$$

A fourth order 2-stage implicit RK method is given by

$$\mathbf{A}_s = \begin{bmatrix} \frac{1}{4} & \frac{1}{24}(6-4\sqrt{3}) \\ \frac{1}{24}(6+4\sqrt{3}) & \frac{1}{4} \end{bmatrix} \qquad \mathbf{b}_s = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix} \quad \text{and} \quad \mathbf{c}_s = \begin{bmatrix} \frac{1}{6}(3-\sqrt{3}) \\ \frac{1}{6}(3+\sqrt{3}) \end{bmatrix},$$

and hence at some time step k we have

$$\mathbf{A}_k = \begin{bmatrix} \mathbf{I}_N - \frac{\tau_k}{4}\mathbf{L} & -\frac{\tau_k}{24}\left(6-4\sqrt{3}\right)\mathbf{L} \\ -\frac{\tau_k}{24}\left(6+4\sqrt{3}\right)\mathbf{L} & \mathbf{I}_N - \frac{\tau_k}{4}\mathbf{L} \end{bmatrix} \quad \text{and} \quad \mathbf{B}_k = -\begin{bmatrix} \frac{\tau_k}{2}\mathbf{I}_N \\ \frac{\tau_k}{2}\mathbf{I}_N \end{bmatrix}.$$

3. The Discrete Adjoint Method. The abstract representation of a generic linear time-stepping scheme (1.4) is now used in conjunction with the adjoint method to systematically find the procedures for computing the action of the sensitivity matrix, as well as the action of the Hessian (and its approximations) of the misfit function $\mathcal{M} = \mathcal{M}(\mathbf{d}; \mathbf{p})$ with respect to the model parameters $\mathbf{p} = \left[\mathbf{m}^\top \mathbf{s}^\top\right]^\top$.

The sensitivity matrix is given by $\mathbf{J} = \frac{\partial \mathbf{d}}{\partial \mathbf{p}} = \frac{\partial \mathbf{d}}{\partial \mathbf{y}} \left[\frac{\partial \mathbf{y}}{\partial \mathbf{m}} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{s}}\right]$ (see (1.2)), where we have used the chain rule, so we need to find expressions for the terms $\frac{\partial \mathbf{y}}{\partial \mathbf{m}}$ and $\frac{\partial \mathbf{y}}{\partial \mathbf{s}}$.

- To find $\frac{\partial \mathbf{y}}{\partial \mathbf{m}}$, we start by differentiating (1.4) with respect to $\mathbf{m}$ on both sides, where the implicit dependence of $\mathbf{y}$ on $\mathbf{m}$ is now included. Then

$$\frac{\partial}{\partial \mathbf{m}}\left(\widehat{\mathbf{T}}(\mathbf{m})\,\mathbf{y}(\mathbf{m}) + \mathbf{S}(\mathbf{m})\,\mathbf{y}_0\right) = \mathbf{0}_{N \times N_\mathbf{m}}$$

$$\Rightarrow \qquad \frac{\partial\left(\widehat{\mathbf{T}}(\mathbf{m})\,\mathbf{y}\right)}{\partial\,\mathbf{m}} + \widehat{\mathbf{T}}\,\frac{\partial\,\mathbf{y}}{\partial\,\mathbf{m}} + \frac{\partial\left(\mathbf{S}(\mathbf{m})\,\mathbf{y}_0\right)}{\partial\,\mathbf{m}} = \mathbf{0}_{N \times N_\mathbf{m}}.$$

We applied the chain and product rules, and the lack of explicit dependence of $\mathbf{y}$ on $\mathbf{m}$ in the first term should be interpreted as $\mathbf{y}$ being fixed with respect to $\mathbf{m}$. It follows that

$$\frac{\partial \mathbf{y}}{\partial \mathbf{m}} = -\widehat{\mathbf{T}}^{-1} \frac{\partial \mathbf{T}}{\partial \mathbf{m}},$$

where $\mathbf{T} = \mathbf{T}(\mathbf{y}, \mathbf{m})$.

- Similarly, suppressing dependence on the field parameters $\mathbf{m}$ and taking the derivative with respect to the source parameters $\mathbf{s}$ on both sides of (1.4) leads to

$$\frac{\partial \mathbf{y}}{\partial \mathbf{s}} = \widehat{\mathbf{T}}^{-1} \frac{\partial \mathbf{q}(\mathbf{s})}{\partial \mathbf{s}}.$$

Hence

$$\mathbf{J} = \frac{\partial \mathbf{d}}{\partial \mathbf{y}} \widehat{\mathbf{T}}^{-1} \left[-\frac{\partial \mathbf{T}}{\partial \mathbf{m}} \quad \frac{\partial \mathbf{q}(\mathbf{s})}{\partial \mathbf{s}} \right],$$

and therefore

$$\mathbf{J}^\top = \begin{bmatrix} -\dfrac{\partial \mathbf{T}}{\partial \mathbf{m}}^\top \\ \dfrac{\partial \mathbf{q}(\mathbf{s})}{\partial \mathbf{s}}^\top \end{bmatrix} \widehat{\mathbf{T}}^{-\top} \frac{\partial \mathbf{d}}{\partial \mathbf{y}}^\top.$$

It is neither desirable nor necessary to compute the Jacobian $\mathbf{J}$ explicitly, what one really needs is to be able to quickly compute the action of $\mathbf{J}$ and its transpose with appropriately-sized arbitrary vectors $\mathbf{w}$. To illustrate how the adjoint solution and the derivatives of $\mathbf{T}$ are used, we summarize the computation of the action of $\mathbf{J}$ in Algorithm 3.1, and that of its transpose in Algorithm 3.2. These algorithms are meant to provide the context for our work in Sections 4 and 5.

ALGORITHM 3.1. Computing the Action of $\mathbf{J}$

Let $\mathbf{w} = \begin{bmatrix} \mathbf{w}_\mathbf{m}^\top & \mathbf{w}_\mathbf{s}^\top \end{bmatrix}^\top$ *be an arbitrary vector of length* $N_\mathbf{m} + N_\mathbf{s}$. *The product* $\mathbf{u} = \mathbf{J}\,\mathbf{w}$ *is computed as follows:*

1. *If the forward solution* $\mathbf{y}$ *is not available already, solve for and store* $\mathbf{y} = \widehat{\mathbf{T}}^{-1}\,(\mathbf{q} - \mathbf{S}\,\mathbf{y}_0)$.
2. *For* $k = 1, \cdots, K$*:*
   (a) *Compute* $\boldsymbol{\theta}_k = \dfrac{\partial \mathbf{q}_k}{\partial \mathbf{s}}\,\mathbf{w}_\mathbf{s} - \dfrac{\partial \mathbf{T}_k}{\partial \mathbf{m}}\,\mathbf{w}_\mathbf{m}$.
   (b) *Solve for* $\mathbf{v}_k$ *using one step of the forward time-stepping method, with* $\boldsymbol{\theta}_k$ *acting as the source term and* $\mathbf{v}_0 = \mathbf{0}_{N\times 1}$.
   (c) *Compute* $\mathbf{u}_k = \dfrac{\partial \mathbf{d}_k}{\partial \mathbf{y}_k}\,\mathbf{v}_k$.
3. *Set* $\mathbf{u} = \begin{bmatrix} \mathbf{u}_1^\top & \cdots & \mathbf{u}_K^\top \end{bmatrix}^\top$.

□

In order to compute $\mathbf{J}^\top\,\mathbf{w}$, we will need to perform the adjoint time-stepping method to get the adjoint solution $\boldsymbol{\lambda} = \widehat{\mathbf{T}}^{-\top} \dfrac{\partial \mathbf{d}}{\partial \mathbf{y}}^\top \mathbf{w}$. As we saw in Section 2, $\widehat{\mathbf{T}}$ is lower block-diagonal and this corresponds to forward integration in time, so $\widehat{\mathbf{T}}^\top$ is block upper triangular and therefore corresponds to backward integration in time.

ALGORITHM 3.2. Computing the Action of $\mathbf{J}^\top$

Let $\mathbf{w} = \begin{bmatrix}\mathbf{w}_1^\top & \cdots & \mathbf{w}_K^\top\end{bmatrix}^\top$ *be an arbitrary vector of length* $N_{\mathbf{d}} = \sum_{k=1}^{K} N_{\mathbf{d}_k}$. *The product* $\mathbf{u} = \mathbf{J}^\top \mathbf{w}$ *is computed as follows:*

1. *Set* $\mathbf{u}_{\mathbf{m}} = \mathbf{0}_{N_{\mathbf{m}}\times 1}$ *and* $\mathbf{u}_s = \mathbf{0}_{N_{\mathbf{s}}\times 1}$. *These will store* $\mathbf{u}$.
2. *If the forward solution* $\mathbf{y}$ *is not available already, solve for and store* $\mathbf{y} = \widehat{\mathbf{T}}^{-1}\left(\mathbf{q} - \mathbf{S}\,\mathbf{y}_0\right)$.
3. *For* $k = K, \cdots, 1$*:*
   (a) *Compute* $\boldsymbol{\theta}_k = \dfrac{\partial\,\mathbf{d}_k}{\partial\,\mathbf{y}_k}^\top \mathbf{w}_k$.
   (b) *Solve for* $\boldsymbol{\lambda}_k$ *using one step of the adjoint time-stepping method, with* $\boldsymbol{\theta}_k$ *acting as the adjoint source term and* $\boldsymbol{\lambda}_{K+1} = \mathbf{0}_{N\times 1}$. *See Algorithm 4.1 if using an LM method and Algorithm 4.2 if using an RK method.*
   (c) *Compute* $\mathbf{u}_{\mathbf{m}} = \mathbf{u}_{\mathbf{m}} - \dfrac{\partial\,\mathbf{T}_k}{\partial\,\mathbf{m}}^\top \boldsymbol{\lambda}_k$ *using (5.1) if using LM and (5.3) if using RK.*
   (d) *Compute* $\mathbf{u}_{\mathbf{s}} = \mathbf{u}_{\mathbf{s}} + \dfrac{\partial\,\mathbf{q}_k}{\partial\,\mathbf{s}}^\top \boldsymbol{\lambda}_k$.
4. *Set* $\mathbf{u} = \begin{bmatrix}\mathbf{u}_{\mathbf{m}}^\top & \mathbf{u}_{\mathbf{s}}^\top\end{bmatrix}^\top$.

□

Note the following:

- The forward solution $\mathbf{y}$ is required to compute both the action of $\mathbf{J}$ and its transpose. In the case of RK methods the forward solution $\mathbf{y}$ includes the internal stages.
- We will need to perform the forward time-stepping method in the computation of $\mathbf{J}\,\mathbf{w}$. This is in addition to the computation of $\mathbf{y}$ (if $\mathbf{y}$ is not already available).
- For RK methods we have slightly abused notation, as $\boldsymbol{\theta}_k$, $\mathbf{v}_k$ and $\boldsymbol{\lambda}_k$ are assumed to also include the internal stages for that time step, therefore these vectors are of length $(s+1)N$. This does not apply to the initial condition $\mathbf{v}_0$ and the final condition $\boldsymbol{\lambda}_{K+1}$, which are both only of length N.
- In the given algorithms we have assumed that the observation $\mathbf{d}_k$ depends only on $\mathbf{y}_k$, i.e. the data at a given point in time depends on the solution only at that time. In case the data depends on the solution over a longer time period, for instance if the data is actually the average of the solution over some time interval, one naturally needs to adapt the computation of $\mathbf{u}_k$ in Algorithm 3.1 and the computation of $\boldsymbol{\theta}_k$ in Algorithm 3.2, although this will be straightforward to do in most cases.

Now, to compute the gradient of the objective function, we use the chain rule to immediately get

$$\nabla_{\mathbf{p}}\mathcal{M} = \frac{\partial\,\mathbf{d}}{\partial\,\mathbf{p}}^\top \nabla_{\mathbf{d}}\mathcal{M} = \mathbf{J}^\top \nabla_{\mathbf{d}}\mathcal{M}.$$

The gradient $\nabla_{\mathbf{p}}\mathcal{M}$ is therefore easily obtained using Algorithm 3.2 with $\mathbf{w} = \nabla_{\mathbf{d}}\mathcal{M}$.

The gradient is used by all gradient-based optimization methods. Newton-type methods are a subset of these methods that additionally require either the action of the full Hessian

$$\mathcal{H}_{\mathcal{M}}\mathbf{w} := \frac{\partial^2\mathcal{M}}{\partial\,\mathbf{p}\,\partial\,\mathbf{p}}\,\mathbf{w} = \nabla_{\mathbf{p}}\left(\nabla_{\mathbf{p}}\mathcal{M}^\top\,\mathbf{w}\right),$$

where $\mathbf{w} = \begin{bmatrix}\mathbf{w}_{\mathbf{m}}^\top & \mathbf{w}_{\mathbf{s}}^\top\end{bmatrix}^\top$ is an arbitrary vector of length $N_{\mathbf{m}} + N_{\mathbf{s}}$, or the action of an approximation of the Hessian that is easier to compute.

The adjoint method can be used to write the action of the full Hessian as

$$\mathcal{H}_{\mathcal{M}}\mathbf{w} = \begin{bmatrix} -\frac{\partial \mathbf{T}}{\partial \mathbf{m}}^{\top} \\ \frac{\partial \mathbf{q}}{\partial \mathbf{s}}^{\top} \end{bmatrix} \boldsymbol{\mu} - \begin{bmatrix} \left(\frac{\partial}{\partial \mathbf{m}}\left(\frac{\partial \mathbf{T}}{\partial \mathbf{m}}\mathbf{w_m}\right)\right)^{\top} + \frac{\partial \widehat{\mathbf{T}}\mathbf{v}}{\partial \mathbf{m}}^{\top} \\ -\left(\frac{\partial}{\partial \mathbf{s}}\left(\frac{\partial \mathbf{q}}{\partial \mathbf{s}}\mathbf{w_s}\right)\right)^{\top} \end{bmatrix} \boldsymbol{\lambda} \tag{3.1a}$$

with $\boldsymbol{\lambda} = \widehat{\mathbf{T}}^{-\top} \frac{\partial \mathbf{d}}{\partial \mathbf{y}}^{\top} \nabla_{\mathbf{d}}\mathcal{M}$, $\mathbf{v} = \widehat{\mathbf{T}}^{-1}\left(\frac{\partial \mathbf{q}}{\partial \mathbf{s}}\mathbf{w_s} - \frac{\partial \mathbf{T}}{\partial \mathbf{m}}\mathbf{w_m}\right)$ and

$$\boldsymbol{\mu} = \widehat{\mathbf{T}}^{-\top}\left(\frac{\partial \mathbf{d}}{\partial \mathbf{y}}^{\top} \frac{\partial^2 \mathcal{M}}{\partial \mathbf{d}\,\partial \mathbf{d}} \frac{\partial \mathbf{d}}{\partial \mathbf{y}}\mathbf{v} + \left(\frac{\partial}{\partial \mathbf{y}}\left(\frac{\partial \mathbf{d}}{\partial \mathbf{y}}\mathbf{v}\right)\right)^{\top} \nabla_{\mathbf{d}}\mathcal{M} - \left(\frac{\partial}{\partial \mathbf{y}}\left(\frac{\partial \mathbf{T}}{\partial \mathbf{m}}\mathbf{w_m}\right)\right)^{\top} \boldsymbol{\lambda}\right). \tag{3.1b}$$

The dependence of $\mathcal{M}$ on both field and source parameters makes the expression more complicated than usual. We are not aware of a derivation of it in the literature, so we have included a detailed derivation in the appendix, but see e.g. [31] for a derivation of the Hessian with respect to only $\mathbf{m}$. Note that the presence of an extra set of parameters $\mathbf{s}$ leads to cross-terms $\frac{\partial^2 \mathcal{M}}{\partial \mathbf{s}\,\partial \mathbf{m}}$ and $\frac{\partial^2 \mathcal{M}}{\partial \mathbf{m}\,\partial \mathbf{s}}$ in the Hessian.

Discussing the implementation details of (3.1) lies outside the scope of this paper, but notice that we require the second derivatives of $\mathbf{T}$. We will discuss how to compute these at a given time step for the time-stepping methods under consideration in Section 5, in case the interested reader might want to attempt the implementation of (3.1).

The full Hessian is actually rarely used in practice though, in large part because of the added expense of computing the second derivatives and the difficulty in coding them. It will also generally not lead to improved convergence rates in numerical minimization procedures unless one is fairly close to the minimum already, so using the full Hessian might not always be beneficial even if an efficient implementation is already available.

As a result, the action of the Hessian is often approximated, for instance in the case of the Gauss-Newton method where we use the symmetric positive semi-definite approximation

$$\mathcal{H}_{\mathcal{M}}\mathbf{w} \approx \mathcal{H}^{\mathrm{GN}}_{\mathcal{M}}\mathbf{w} = \mathbf{J}^{\top} \frac{\partial^2 \mathcal{M}}{\partial \mathbf{d}\,\partial \mathbf{d}}\,\mathbf{J}\,\mathbf{w},$$

which is obtained by omitting all the second derivatives of $\mathbf{T}$ and $\mathbf{d}$. A related method is the Levenberg-Marquardt method, where we add a damping term $\beta > 0$ to get a symmetric positive definite approximation

$$\mathcal{H}_{\mathcal{M}}\mathbf{w} \approx \mathcal{H}^{\mathrm{LM}}_{\mathcal{M}}\mathbf{w} = \left(\mathbf{J}^{\top} \frac{\partial^2 \mathcal{M}}{\partial \mathbf{d}\,\partial \mathbf{d}}\,\mathbf{J} + \beta\,\mathbf{I}\right)\mathbf{w}.$$

Both of these approximations are computed using Algorithm 3.1 followed by Algorithm 3.2, so that in addition to $\mathbf{y}$ we will have to compute one forward solution and one adjoint solution.

The full Hessian also requires a forward solution $\mathbf{v}$ in addition to $\mathbf{y}$, and $\mathbf{v}$ will have to be stored. Two adjoint solutions are needed, neither of which need to be stored in full, although LM methods will need access to the s previous steps. Note that a Newton-type method generally will also require the computation of the gradient and consequently $\boldsymbol{\lambda}$ will already have been computed, so if using the full Hessian and if enough storage space is available one can store $\boldsymbol{\lambda}$ and thereby reduce the cost of the overall computation. Computing the full Hessian in this case requires as many forward and adjoint solves as the approximations discussed above. The cost of storing one extra forward solution can be prohibitive in many circumstances though.

4. Solving the Adjoint Problem. One major ingredient of the adjoint method is the solution of the adjoint problem,

$$\boldsymbol{\lambda} = \widehat{\mathbf{T}}^{-\top}\,\boldsymbol{\theta},$$

where $\widehat{\mathbf{T}} = \widehat{\mathbf{T}}(\mathbf{m})$[2] is either the LM or the RK time-stepping matrix (defined in Sections 2.1 and 2.2, respectively), $\boldsymbol{\lambda} = \begin{bmatrix}\boldsymbol{\lambda}_1^\top & \dots & \boldsymbol{\lambda}_K^\top\end{bmatrix}$ is the adjoint solution, and $\boldsymbol{\theta} = \begin{bmatrix}\boldsymbol{\theta}_1^\top & \dots & \boldsymbol{\theta}_K^\top\end{bmatrix}$ is taken in this section to be some arbitrary adjoint source. For RK methods we implicitly include the intermediate stages in $\boldsymbol{\lambda}_k$ and $\boldsymbol{\theta}_k$. The time-stepping matrices are lower block-triangular and therefore the forward problem can conceptually be seen as being solved by forward substitution, which corresponds to solving the problem forward in time. The adjoint of $\widehat{\mathbf{T}}$, i.e. its transpose in the finite-dimensional setting we are in, will thus be upper block-triangular and is therefore solved by backward substitution, which corresponds to solving the adjoint problem backward in time.

4.1. Multistep Methods. Since $\widehat{\mathbf{T}} = \mathbf{A} - \tau\,\mathbf{B}\,\mathrm{diag}(\mathbf{L})$, we have $\widehat{\mathbf{T}}^\top = \mathbf{A}^\top - \tau\,\mathrm{diag}(\mathbf{L}^\top)\,\mathbf{B}^\top$. See Section 2.1 for the definitions of $\mathbf{A}$ and $\mathbf{B}$, and also keep the structure of the template matrices defined in that section in mind. Setting $\mathbf{C}_j^{(\sigma)} = \alpha_j^{(\sigma)}\mathbf{I} - \tau\,\beta_j^{(\sigma)}\mathbf{L}$, $\mathbf{C}_j = \mathbf{C}_j^{(s)}$, and

$$\widehat{\mathbf{T}}^{(1:s)} = \begin{bmatrix} \mathbf{C}_0^{(1)} & & \\ \vdots & \ddots & \\ \mathbf{C}_{s-2}^{(s-1)} & \dots & \mathbf{C}_0^{(s-1)} \end{bmatrix},$$

the adjoint problem has the following structure:

$$\underbrace{\begin{bmatrix} \widehat{\mathbf{T}}^{(1:s)\top} & \mathbf{C}_{s-1}^\top & \mathbf{C}_s^\top & & & & & \\ & \vdots & \ddots & \ddots & & & & \\ & \mathbf{C}_1^\top & & \ddots & \ddots & & & \\ & \mathbf{C}_0^\top & \mathbf{C}_1^\top & \cdots & \mathbf{C}_{s-1}^\top & \mathbf{C}_s^\top & & \\ & & \ddots & \ddots & & \ddots & \ddots & \\ & & & \ddots & \ddots & & \ddots & \mathbf{C}_s^\top \\ & & & & \ddots & \ddots & & \mathbf{C}_{s-1}^\top \\ & & & & & \ddots & \ddots & \vdots \\ & & & & & & \mathbf{C}_0^\top & \mathbf{C}_1^\top \\ & & & & & & & \mathbf{C}_0^\top \end{bmatrix}}_{\widehat{\mathbf{T}}^\top} \underbrace{\begin{bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ \vdots \\ \boldsymbol{\lambda}_{s-1} \\ \boldsymbol{\lambda}_s \\ \boldsymbol{\lambda}_{s+1} \\ \vdots \\ \vdots \\ \boldsymbol{\lambda}_{K-1} \\ \boldsymbol{\lambda}_K \end{bmatrix}}_{\boldsymbol{\lambda}} = \underbrace{\begin{bmatrix} \boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2 \\ \vdots \\ \boldsymbol{\theta}_{s-1} \\ \boldsymbol{\theta}_s \\ \boldsymbol{\theta}_{s+1} \\ \vdots \\ \vdots \\ \boldsymbol{\theta}_{K-1} \\ \boldsymbol{\theta}_K \end{bmatrix}}_{\boldsymbol{\theta}}.$$

In iterative form the backward substitution becomes

$$\text{for } s \le k \le K: \qquad \mathbf{C}_0^\top\,\boldsymbol{\lambda}_k = \boldsymbol{\theta}_k - \sum_{j=1}^{\min(s,K-k)} \mathbf{C}_j^\top\,\boldsymbol{\lambda}_{k+j}$$

$$\text{for } 1 \le k < s: \qquad \mathbf{C}_0^{(k)\top}\,\boldsymbol{\lambda}_k = \boldsymbol{\theta}_k - \sum_{j=1}^{s-k-1} \mathbf{C}_j^{(k+j)\top}\,\boldsymbol{\lambda}_{k+j} - \sum_{j=s-k}^{\min(s,K-k)} \mathbf{C}_j^\top\,\boldsymbol{\lambda}_{k+j}.$$

[2] In this section the explicit dependence on $\mathbf{m}$ is again omitted to make the notation less cumbersome.

From this the procedure for finding the adjoint solution is found and is summarized in Algorithm 4.1.

ALGORITHM 4.1. The Adjoint Linear Multistep Time-Stepping Method

The method is explicit if $\beta_0^{(\sigma)} = 0$*, implicit if* $\beta_0^{(\sigma)} \neq 0$.

1. *For* $k = K, \cdots, s$ *solve*

$$(\mathbf{I} - \tau\, \beta_0^{(s)} \mathbf{L}^\top)\, \boldsymbol{\lambda}_k = \boldsymbol{\theta}_k - \sum_{j=1}^{\min(s,K-k)} (\alpha_j^{(s)} \mathbf{I} - \tau\, \beta_j^{(s)} \mathbf{L}^\top)\, \boldsymbol{\lambda}_{k+j}.$$

2. *For* $k = s-1, \cdots, 1$*, solve*

$$(\mathbf{I} - \tau\, \beta_0^{(k)} \mathbf{L}^\top)\, \boldsymbol{\lambda}_k = \boldsymbol{\theta}_k - \sum_{j=1}^{s-k-1} (\alpha_j^{(k+j)} \mathbf{I} - \tau\, \beta_j^{(k+j)} \mathbf{L}^\top)\, \boldsymbol{\lambda}_{k+j} - \sum_{j=s-k}^{\min(s,K-k)} (\alpha_j^{(s)} \mathbf{I} - \tau\, \beta_j^{(s)} \mathbf{L}^\top)\, \boldsymbol{\lambda}_{k+j}. \qquad \square$$

We make the following observations:

- For explicit methods $\boldsymbol{\lambda}_K = \boldsymbol{\theta}_K$.
- While the adjoint LM method may look like it is essentially the same as its corresponding forward method, the source terms are actually handled differently. For the forward method the source terms are a linear combination of q at different times (with the particular LM method determining the coefficients of this combination), whereas this is not the case in the adjoint method, where the $\boldsymbol{\theta}_k$ are used as they are.
- Apart from the source terms, the first few steps of the adjoint s-step method are essentially the same as the usual s-step method with $\boldsymbol{\lambda}_k = \mathbf{0}_N$ for all $k > K$.
- Finally, while the adjoint method is practically the same as the forward LM method (apart from the source terms), the last few steps $k = 1, \cdots, s-1$ introduce a mix of coefficients from the s-step method and the lower order methods that were used during the initialization steps of the forward method, possibly making the final couple of steps of the adjoint method unstable or inconsistent. While the initialization steps are important for the forward problem, they are more likely to do more harm than good in the adjoint method. It might therefore be sensible in practice to just use the s-step method all the way through, although it should not make a difference either way since these steps are few in number and occur at the end of the integration.

Example

The first couple of iterations of the adjoint three-step Adams-Bashforth method are

$$\begin{aligned}
\boldsymbol{\lambda}_K &= \boldsymbol{\theta}_K \\
\boldsymbol{\lambda}_{K-1} &= \boldsymbol{\theta}_{K-1} + (\mathbf{I} + \tau\, \tfrac{23}{12}\, \mathbf{L}^\top)\boldsymbol{\lambda}_K \\
\boldsymbol{\lambda}_{K-2} &= \boldsymbol{\theta}_{K-2} + (\mathbf{I} + \tau\, \tfrac{23}{12}\, \mathbf{L}^\top)\boldsymbol{\lambda}_{K-1} - \tau\, \tfrac{16}{12}\, \mathbf{L}^\top \boldsymbol{\lambda}_K \\
\boldsymbol{\lambda}_{K-3} &= \boldsymbol{\theta}_{K-3} + (\mathbf{I} + \tau\, \tfrac{23}{12}\, \mathbf{L}^\top)\boldsymbol{\lambda}_{K-2} - \tau\, \tfrac{16}{12}\, \mathbf{L}^\top \boldsymbol{\lambda}_{K-1} + \tau\, \tfrac{5}{12}\, \mathbf{L}^\top \boldsymbol{\lambda}_K.
\end{aligned}$$

For $k = K-3, \cdots, 2$:

$$\boldsymbol{\lambda}_k = \boldsymbol{\theta}_k + (\mathbf{I} + \tau\, \tfrac{23}{12}\, \mathbf{L}^\top)\boldsymbol{\lambda}_{k+1} - \tau\, \tfrac{16}{12}\, \mathbf{L}^\top \boldsymbol{\lambda}_{k+2} + \tau\, \tfrac{5}{12}\, \mathbf{L}^\top \boldsymbol{\lambda}_{k+3}.$$

For $k = 1$:

$$\boldsymbol{\lambda}_1 = \boldsymbol{\theta}_1 + (\mathbf{I} + \tau\, \tfrac{3}{2}\, \mathbf{L}^\top)\boldsymbol{\lambda}_2 - \tau\, \tfrac{16}{12}\, \mathbf{L}^\top \boldsymbol{\lambda}_3 + \tau\, \tfrac{5}{12}\, \mathbf{L}^\top \boldsymbol{\lambda}_4.$$

4.2. Runge-Kutta Methods. The RK time-stepping system is given in (2.6). From this it follows that the general adjoint problem $\boldsymbol{\lambda} = \widehat{\mathbf{T}}^{-1}\boldsymbol{\theta}$ has, for RK methods, the following structure:

$$\begin{bmatrix} \mathbf{A}_1^\top & \mathbf{B}_1 & & & & & & \\ \mathbf{I}_N & \mathbf{D}^\top & -\mathbf{I}_N & & & & & \\ & \mathbf{A}_2^\top & \mathbf{B}_2 & & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \mathbf{I}_N & \mathbf{D}^\top & -\mathbf{I}_N & & \\ & & & & \mathbf{A}_K^\top & \mathbf{B}_K & & \\ & & & & & \mathbf{I}_N & & \end{bmatrix} \begin{bmatrix} \boldsymbol{\Lambda}_1 \\ \boldsymbol{\lambda}_1 \\ \boldsymbol{\Lambda}_2 \\ \vdots \\ \boldsymbol{\lambda}_{K-1} \\ \boldsymbol{\Lambda}_K \\ \boldsymbol{\lambda}_K \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Theta}_1 \\ \boldsymbol{\theta}_1 \\ \boldsymbol{\Theta}_2 \\ \vdots \\ \boldsymbol{\theta}_{K-1} \\ \boldsymbol{\Theta}_K \\ \boldsymbol{\theta}_K \end{bmatrix}$$

where $\boldsymbol{\lambda}_k$ is the adjoint solution and $\boldsymbol{\Lambda}_k = \begin{bmatrix} \boldsymbol{\Lambda}_{k,1}^\top & \cdots & \boldsymbol{\Lambda}_{k,s}^\top \end{bmatrix}^\top$ are the associated internal stages, and $\boldsymbol{\theta}_k$ is the adjoint source at the kth time step with internal stages $\boldsymbol{\Theta}_k = \begin{bmatrix} \boldsymbol{\Theta}_{k,1}^\top & \cdots & \boldsymbol{\Theta}_{k,s}^\top \end{bmatrix}^\top$. The internal stages $\boldsymbol{\Theta}_{k,\sigma}$ of the adjoint source term will be $\mathbf{0}$ in practice (cf. Algorithm 3.2 and use the fact that the data will not directly depend on the internal stages of the numerical method), so we will ignore them from now on.

Recall that $\mathbf{D} = -\mathbf{I}_s \otimes \mathbf{L}$, $\mathbf{A} = \mathbf{I}_{sN} - \tau_k \mathbf{A}_s \otimes \mathbf{L}$ and $\mathbf{B} = -\tau_k \mathbf{b}_s \otimes \mathbf{I}_N$ (see Section 2.2) and that $\mathbf{A}_s$ is the $s \times s$ matrix containing the RK coefficients and $\mathbf{b}_s$ is the vector of length s containing the weights.

The upper block-triangular structure means that we solve this system using backsubstitution, implying that the adjoint solution is solved backward in time. For a given time step $1 \le k \le K$ we start with the update formula

$$\boldsymbol{\lambda}_k = \boldsymbol{\theta}_{k+1} + \boldsymbol{\lambda}_{k+1} - \mathbf{D}^\top \boldsymbol{\Lambda}_{k+1},$$

with $\boldsymbol{\lambda}_{K+1} = \boldsymbol{\theta}_{K+1} = \mathbf{0}_{N\times 1}$. This is followed by the internal adjoint stages

$$\begin{aligned} \mathbf{A}^\top \boldsymbol{\Lambda}_k &= -\mathbf{B}\, \boldsymbol{\lambda}_k \\ \Rightarrow \qquad \left(\mathbf{I}_{sN} - \tau_k \mathbf{A}_s^\top \otimes \mathbf{L}^\top\right) \boldsymbol{\Lambda}_k &= \tau_k \left(\mathbf{b}_s \otimes \mathbf{I}_N\right) \boldsymbol{\lambda}_k. \end{aligned}$$

The cases of explicit, diagonally implicit and fully implicit RK schemes are summarized in Algorithm 4.2.

ALGORITHM 4.2. The Adjoint Runge-Kutta Time-Stepping Method

For $k = K, \cdots, 1$, *compute*

$$\boldsymbol{\lambda}_k = \boldsymbol{\theta}_k + \boldsymbol{\lambda}_{k+1} + \mathbf{L}^\top \sum_{\sigma=1}^{s} \boldsymbol{\Lambda}_{k+1,\sigma}.$$

with $\boldsymbol{\lambda}_{K+1} = \boldsymbol{\Lambda}_{K+1,\sigma} = \mathbf{0}_{N\times 1}$, *then for* $\sigma = s, \cdots, 1$ *compute*

$$\boldsymbol{\Lambda}_{k,\sigma} = \tau_k b_\sigma \boldsymbol{\lambda}_k + \tau_k \mathbf{L}^\top \sum_{i=1}^{s} a_{i\sigma} \boldsymbol{\Lambda}_{k,i}.$$

□

Notice that the formula for $\boldsymbol{\lambda}_k$ is independent of the RK coefficients and weights, and that the adjoint source appears only in the update for $\boldsymbol{\lambda}_k$ and not in the internal stages. How the internal stages are computed depends on if the method itself is explicit, diagonally implicit or fully implicit:

- if the method is *explicit*, for $\sigma = s, \cdots, 1$ compute:

$$\boldsymbol{\Lambda}_{k,\sigma} = \tau_k b_\sigma \boldsymbol{\lambda}_k + \tau_k \mathbf{L}^\top \sum_{i=\sigma+1}^{s} a_{i\sigma} \boldsymbol{\Lambda}_{k,i}$$

- if the method is *diagonally implicit*, for $i = s, \cdots, 1$ solve:

$$\left(\mathbf{I}_N - \tau_k a_{\sigma\sigma} \mathbf{L}^\top\right) \boldsymbol{\Lambda}_{k,\sigma} = \tau_k b_\sigma \boldsymbol{\lambda}_k + \tau_k \mathbf{L}^\top \sum_{i=\sigma+1}^{s} a_{i\sigma} \boldsymbol{\Lambda}_{k,i}$$

- if the method is *fully implicit*, solve:

$$\left(\mathbf{I}_{sN} - \tau_k \mathbf{A}_s^\top \otimes \mathbf{L}^\top\right) \boldsymbol{\Lambda}_k = \tau_k \mathbf{b}_s \otimes \boldsymbol{\lambda}_k.$$

Examples

The adjoint RK4 method is, for $k = K, \cdots, 1$:

$$\boldsymbol{\lambda}_k = \boldsymbol{\theta}_k + \boldsymbol{\lambda}_{k+1} + \mathbf{L}^\top \left(\boldsymbol{\Lambda}_{k+1,1} + \boldsymbol{\Lambda}_{k+1,2} + \boldsymbol{\Lambda}_{k+1,3} + \boldsymbol{\Lambda}_{k+1,4}\right)$$

followed by

$$\begin{aligned}
\boldsymbol{\Lambda}_{k,4} &= \frac{\tau_k}{6} \boldsymbol{\lambda}_k \\
\boldsymbol{\Lambda}_{k,3} &= \frac{\tau_k}{3} \boldsymbol{\lambda}_k + \tau_k \mathbf{L}^\top \boldsymbol{\Lambda}_{k,4} \\
\boldsymbol{\Lambda}_{k,2} &= \frac{\tau_k}{3} \boldsymbol{\lambda}_k + \frac{\tau_k}{2} \mathbf{L}^\top \boldsymbol{\Lambda}_{k,3} \\
\boldsymbol{\Lambda}_{k,1} &= \frac{\tau_k}{6} \boldsymbol{\lambda}_k + \frac{\tau_k}{2} \mathbf{L}^\top \boldsymbol{\Lambda}_{k,2}
\end{aligned}$$

$\boldsymbol{\lambda}_k$ should of course be updated as soon as each product $\mathbf{L}^\top \boldsymbol{\Lambda}_{k+1,\sigma}$ has been computed.

The adjoint method corresponding to the 2-stage implicit RK method given in Section 2.2 is

$$\boldsymbol{\lambda}_k = \boldsymbol{\theta}_k + \boldsymbol{\lambda}_{k+1} + \mathbf{L}^\top \left(\boldsymbol{\Lambda}_{k+1,1} + \boldsymbol{\Lambda}_{k+1,2}\right)$$

with internal stages

$$\begin{aligned}
\boldsymbol{\Lambda}_{k,2} &= \frac{\tau_k}{2} \boldsymbol{\lambda}_k + \frac{\tau_k}{24}(6 - 4\sqrt{3})\, \mathbf{L}^\top \boldsymbol{\Lambda}_{k,1} + \frac{\tau_k}{4} \mathbf{L}^\top \boldsymbol{\Lambda}_{k,2} \\
\boldsymbol{\Lambda}_{k,1} &= \frac{\tau_k}{2} \boldsymbol{\lambda}_k + \frac{\tau_k}{4} \mathbf{L}^\top \boldsymbol{\Lambda}_{k,1} + \frac{\tau_k}{24}(6 + 4\sqrt{3})\, \mathbf{L}^\top \boldsymbol{\Lambda}_{k,2}.
\end{aligned}$$

4.3. Stability,Convergence, and Order of Accuracy. Let us briefly discuss some properties of interest of the adjoint RK methods. A much more detailed analysis of the order of accuracy of these methods (up to order 4) has been given for nonlinear problems in [17], but

we find that for linear problems the simple argument given here should suffice, which we summarize in the following theorem:

THEOREM 4.3. *For linear PDEs, the adjoint RK method inherits the convergence and stability properties from the corresponding forward method. It also has the same order of accuracy.*

Proof. A single step of any RK method applied to a homogeneous problem can be written as

$$\mathbf{y}^k = \Psi\left(\tau_k \mathbf{L}\right) \mathbf{y}^{k-1}, \tag{4.1}$$

and it is not too hard to show that for explicit methods we have

$$\Psi\left(\tau_k \mathbf{L}\right) = \mathbf{I} + \sum_{\sigma=1}^{s} b_\sigma \mathbf{C}_{k,\sigma}\left(\tau_k \mathbf{L}\right)$$

with

$$\mathbf{C}_{k,\sigma}\left(\tau_k \mathbf{L}\right) = \tau_k \mathbf{L}\left(\mathbf{I}_N + \tau_k \sum_{\sigma_1=1}^{\sigma-1} a_{\sigma\sigma_1}\mathbf{L} + \tau_k^2 \sum_{\sigma_1=2}^{\sigma-1}\sum_{\sigma_2=1}^{\sigma_1-1} a_{\sigma\sigma_1}a_{\sigma_1\sigma_2}\mathbf{L}^2 + \right.$$
$$\left. + \tau_k^3 \sum_{\sigma_1=3}^{\sigma-1}\sum_{\sigma_2=2}^{\sigma_1-1}\sum_{\sigma_3=1}^{\sigma_2-1} a_{\sigma\sigma_1}a_{\sigma_1\sigma_2}a_{\sigma_2\sigma_3}\mathbf{L}^3 + \cdots + \tau_k^{\sigma-1}\prod_{i=2}^{\sigma}(a_{i,i-1}\mathbf{L})\right).$$

It is possible to find an expression similar to this for implicit methods.

Consequently, letting $\Psi_k = \Psi\left(\tau_k \mathbf{L}\right)$, the homogeneous forward problem can be written as

$$\begin{bmatrix} \mathbf{I}_N & & & & \\ -\Psi_2 & \mathbf{I}_N & & & \\ & -\Psi_3 & \mathbf{I}_N & & \\ & & \ddots & \ddots & \\ & & & -\Psi_K & \mathbf{I}_N \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \mathbf{y}_3 \\ \vdots \\ \mathbf{y}_K \end{bmatrix} = \mathbf{S}\,\mathbf{y}_0,$$

and therefore the homogeneous adjoint system is

$$\begin{bmatrix} \mathbf{I}_N & -\Psi_2^\top & & & \\ & \mathbf{I}_N & -\Psi_3^\top & & \\ & & \ddots & \ddots & \\ & & & \mathbf{I}_N & -\Psi_K^\top \\ & & & & \mathbf{I}_N \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ \vdots \\ \boldsymbol{\lambda}_{K-1} \\ \boldsymbol{\lambda}_K \end{bmatrix} = \mathbf{0}.$$

Hence $\boldsymbol{\lambda}_{k-1} = \Psi_k^\top\left(\mathbf{L}\right)\boldsymbol{\lambda}_k$. It is fairly straightforward, albeit tedious, to show that this condition holds for the adjoint RK method we found above. Note that $\Psi_k^\top\left(\mathbf{L}\right) = \Psi_k\left(\mathbf{L}^\top\right)$.

By the Lax equivalence theorem, a consistent linear method in the form (4.1) is convergent if and only if it is Lax-Richtmyer stable. Letting $\tau_k = \tau$ for simplicity, Lax-Richtmyer stability means that for each time T, there is a constant $C_T > 0$ such that

$$\left\|\Psi\left(\tau\mathbf{L}\right)^K\right\| \leq C_T$$

for all $\tau > 0$ and integers K for which $\tau K < T$. We know that this must hold for $\Psi\left(\tau\mathbf{L}\right)$ (because the forward method is stable), and hence the adjoint time-stepping method is Lax-Richtmyer stable as well since matrix transposition is invariant under, e.g., the 2-norm.

Now consider an arbitrary RK method applied to the scalar test problem $\dot{y}(t) = \mu\, y(t)$, where μ is some scalar (it represents one of the eigenvalues of $\mathbf{L}$). Then we have that over a single time step $y_k = \Psi(z)\, y_{k-1}$, where $z = \tau_k \mu$, and if the method is pth order accurate we must have that

$$\Psi(z) - e^z = O(z^{p+1}).$$

Since the same Ψ applies to the corresponding scalar adjoint scheme, it follows that for a single time step the same condition must hold and therefore the adjoint RK method must have the same order of accuracy as the forward method. □

We will not discuss here the order reduction phenomenon [6, 28, 34] that can happen close to the boundary when RK methods are applied to nonhomogeneous problems, but it is possible that the absence of source terms in the internal stages of the adjoint RK methods would lessen this effect. At the least it seems reasonable to assume that the reduction of order suffered by the adjoint RK method would not be worse than that of the RK method, but a more careful analysis needs to be done to support this statement.

Incidentally, it is not hard to show that for the RK4 method we have $\Psi(z) = 1 + z + z^2/2 + z^3/6 + z^4/24$ and for the 2-stage implicit method we have $\Psi(z) = \dfrac{1 + z/2 + z^2/12}{1 - z/2 + z^2/12}$, the latter being a fourth order Padé approximation of the exponential function.

We can apply a similar analysis to LM methods, although in this case it is immediate from the adjoint time-stepping method that, ignoring the source terms and the initialization steps, it is equivalent to the corresponding forward time-stepping method (just relabel $\boldsymbol{\lambda}_k$ and $\boldsymbol{\lambda}_{k-1}$ with $\mathbf{y}_k$ and $\mathbf{y}_{k+1}$ respectively).

5. The derivatives of $\mathbf{T}$. We now focus on the derivatives of $\mathbf{T}(\mathbf{y}, \mathbf{m})$ with respect to $\mathbf{m}$ and $\mathbf{y}$ for a given time step k. We will make reference throughout this section to the derivatives of the products of $\mathbf{L}(\mathbf{m})$ with some arbitrary vectors of length N. Not much can be said about these derivatives in general since $\mathbf{L}$ depends on the PDE, but in the next section we show how to compute the derivatives for the acoustic wave equation.

Since we are actually interested in the action of these derivatives and their transposes, we will give formulas for computing the products of the derivatives with some appropriately-sized arbitrary vectors.

5.1. Linear Multistep Methods. From Section 2.1 we know that the kth time step for LM methods can be represented by $\mathbf{T}_k(\mathbf{y}, \mathbf{m})$. Recalling the way we constructed $\mathbf{T}$ and letting $\mathbf{C}_j^{(\sigma)} = \mathbf{C}_j^{(\sigma)}(\mathbf{m}) = \alpha_j^{(\sigma)}\mathbf{I} - \tau\, \beta_j^{(\sigma)}\mathbf{L}(\mathbf{m})$, $\mathbf{C}_j = \mathbf{C}_j^{(s)}$ and $s^* = \min(k, s)$, we have

$$\mathbf{T}_k(\mathbf{y}, \mathbf{m}) = \sum_{j=0}^{s^*} \mathbf{C}_j^{(s^*)}(\mathbf{m})\, \mathbf{y}_{k-j} = \sum_{j=0}^{s^*} \alpha_j^{(s^*)} \mathbf{y}_{k-j} - \tau \sum_{j=0}^{s^*} \beta_j^{(s^*)}\, \mathbf{L}(\mathbf{m})\, \mathbf{y}_{k-j}.$$

In what follows we let $\mathbf{w}_{\mathbf{m}}$ denote an arbitrary vector of length $N_{\mathbf{m}}$, $\mathbf{w} = \begin{bmatrix}\mathbf{w}_1^\top & \cdots & \mathbf{w}_K^\top\end{bmatrix}^\top$ an arbitrary vector of length KN. The derivatives of this term are trivial.

Taking the derivative of $\mathbf{T}_k$ with respect to $\mathbf{m}$ gives

$$\frac{\partial\, \mathbf{T}_k}{\partial\, \mathbf{m}} = -\tau \sum_{j=0}^{s^*} \beta_j^{(s^*)} \frac{\partial\, (\mathbf{L}\mathbf{y}_{k-j})}{\partial\, \mathbf{m}}, \tag{5.1}$$

then multiplying this by $\mathbf{w_m}$ and taking the derivative with respect to $\mathbf{m}$ gives

$$\frac{\partial^2 \mathbf{T}_k}{\partial \mathbf{m}\, \partial \mathbf{m}} \mathbf{w_m} = -\tau \sum_{j=0}^{s^*} \beta_j^{(s^*)} \frac{\partial^2 \left(\mathbf{L}\mathbf{y}_{k-j}\right)}{\partial \mathbf{m}\, \partial \mathbf{m}} \mathbf{w}_k.$$

Differentiating $\dfrac{\partial \mathbf{T}_k}{\partial \mathbf{m}} \mathbf{w_m}$ with respect to $\mathbf{y}$ instead gives

$$\frac{\partial^2 \mathbf{T}_k}{\partial \mathbf{y}\, \partial \mathbf{m}} \mathbf{w_m} = \begin{bmatrix} \dfrac{\partial^2 \mathbf{T}_k}{\partial \mathbf{y}_1\, \partial \mathbf{m}} \mathbf{w_m} & \cdots & \dfrac{\partial^2 \mathbf{T}_k}{\partial \mathbf{y}_K\, \partial \mathbf{m}} \mathbf{w_m} \end{bmatrix}$$

where

$$\frac{\partial^2 \mathbf{T}_k}{\partial \mathbf{y}_i\, \partial \mathbf{m}} \mathbf{w_m} = \begin{cases} -\tau\, \beta_{k+i}^{(s^*)} \dfrac{\partial^2 \left(\mathbf{L}\mathbf{y}_i\right)}{\partial \mathbf{y}_i\, \partial \mathbf{m}} \mathbf{w}_k & \text{if} \quad \max(1, k - s^*) \le i \le k \\ \mathbf{0}_{N \times N} & \text{otherwise.} \end{cases}$$

Finally,

$$\mathbf{v} = \widehat{\mathbf{T}}(\mathbf{m})\mathbf{w} = \sum_{j=0}^{\min(k-1,s)} \alpha_j^{(s^*)} \mathbf{w}_{k-j} - \tau \sum_{j=0}^{\min(k-1,s)} \beta_j^{(s^*)}\, \mathbf{L}(\mathbf{m})\, \mathbf{w}_{k-j}.$$

Then the derivative with respect to $\mathbf{m}$ of the kth step of this product is just

$$\frac{\partial\, \mathbf{v}_k}{\partial\, \mathbf{m}} = -\tau \sum_{j=0}^{\min(k-1,s)} \beta_j^{(s^*)} \frac{\partial \left(\mathbf{L}\mathbf{w}_{k-j}\right)}{\partial\, \mathbf{m}}.$$

The products of all of these derivatives with appropriately-size arbitrary vectors are clear.

5.2. Runge-Kutta Methods. The derivatives arising from RK methods are more interesting than the ones for LM methods, although still easy to arrive at. As is clear from Section 2.2, a single time step k of a general Runge-Kutta time-stepping method can be represented by, for $1 \le k \le K$,

$$\mathbf{T}_k = \begin{bmatrix} \mathbf{D}\, \mathbf{y}_{k-1} + \mathbf{A}_k\, \mathbf{Y}_k \\ -\mathbf{y}_{k-1} + \mathbf{B}_k^\top\, \mathbf{Y}_k + \mathbf{y}_k \end{bmatrix}$$

so that $\mathbf{T}_{k,\sigma}$ is

$$\mathbf{T}_{k,\sigma} = \begin{cases} \mathbf{y}_k - \mathbf{L}(\mathbf{m}) \underline{\mathbf{y}}_{k,\sigma} + \mathbf{Y}_{k,\sigma} & \text{if } 1 \le \sigma \le s \\ \mathbf{y}_k - \mathbf{y}_{k-1} - \tau_k \sum_{\sigma=1}^{s} b_\sigma\, \mathbf{Y}_{k,\sigma} & \text{if } \sigma = s, \end{cases}$$

where we have let $\underline{\mathbf{y}}_{k,\sigma} = \mathbf{y}_{k-1} + \tau_k \sum_{i=1}^{s} a_{\sigma,i}\, \mathbf{Y}_{k,i}$. In what follows we let $\mathbf{w_m}$ denote an arbitrary vector of length $N_\mathbf{m}$ and $\mathbf{w} = \begin{bmatrix} \widehat{\mathbf{w}}_1^\top & \cdots & \widehat{\mathbf{w}}_K^\top \end{bmatrix}^\top$ an arbitrary vector of length $(s+1)NK$, where $\widehat{\mathbf{w}}_k = \begin{bmatrix} \mathbf{W}_{k,1}^\top & \cdots & \mathbf{W}_{k,s}^\top & \mathbf{w}_k^\top \end{bmatrix}^\top$ are each of length $(s+1)N$. Notice that we require access to the internal stages of the forward solution, so these will have to be stored. Although also notice that in the case of explicit RK methods we do not need the final internal stage.

- $\dfrac{\partial \mathbf{T}}{\partial \mathbf{m}}$:
Taking the derivative of $\mathbf{T}_{k,\sigma}$ with respect to $\mathbf{m}$:

$$\frac{\partial \mathbf{T}_{k,\sigma}}{\partial \mathbf{m}} = \begin{cases} -\dfrac{\partial \mathbf{L}\underline{\mathbf{y}}_{k,\sigma}}{\partial \mathbf{m}} & \text{if } 1 \le \sigma \le s \\ \mathbf{0}_{N \times N_{\mathbf{m}}} & \text{if } \sigma = s. \end{cases}$$

Hence

$$\frac{\partial \mathbf{T}_k}{\partial \mathbf{m}} \mathbf{w}_{\mathbf{m}} = \begin{bmatrix} -\dfrac{\partial \mathbf{L}\underline{\mathbf{y}}_{k,1}}{\partial \mathbf{m}} \mathbf{w}_{\mathbf{m}} \\ \vdots \\ -\dfrac{\partial \mathbf{L}\underline{\mathbf{y}}_{k,s}}{\partial \mathbf{m}} \mathbf{w}_{\mathbf{m}} \\ \mathbf{0}_{N \times 1} \end{bmatrix} \tag{5.2}$$

and

$$\frac{\partial \mathbf{T}_k}{\partial \mathbf{m}}^{\top} \widehat{\mathbf{w}}_k = -\sum_{\sigma=1}^{s} \frac{\partial \mathbf{L}\underline{\mathbf{y}}_{k,\sigma}}{\partial \mathbf{m}}^{\top} \mathbf{W}_{k,\sigma}. \tag{5.3}$$

- $\dfrac{\partial^2 \mathbf{T}}{\partial \mathbf{m} \partial \mathbf{m}} \mathbf{w}_{\mathbf{m}}$:
Taking the derivative of (5.2) with respect to $\mathbf{m}$ immediately gives

$$\frac{\partial^2 \mathbf{T}_k}{\partial \mathbf{m} \partial \mathbf{m}} \mathbf{w}_{\mathbf{m}} = \begin{bmatrix} -\dfrac{\partial^2 \mathbf{L}\underline{\mathbf{y}}_{k,1}}{\partial \mathbf{m} \partial \mathbf{m}} \mathbf{w}_{\mathbf{m}} \\ \vdots \\ -\dfrac{\partial^2 \mathbf{L}\underline{\mathbf{y}}_{k,s}}{\partial \mathbf{m} \partial \mathbf{m}} \mathbf{w}_{\mathbf{m}} \\ \mathbf{0}_{N \times 1} \end{bmatrix}.$$

The products of this $(s+1)N \times N_{\mathbf{m}}$ matrix and its transpose are obvious.

- $\dfrac{\partial^2 \mathbf{T}}{\partial \mathbf{y} \partial \mathbf{m}} \mathbf{w}_{\mathbf{m}}$:
Differentiating (5.2) with respect to $\mathbf{y}$ gives

$$\frac{\partial^2 \mathbf{T}_k}{\partial \mathbf{y} \partial \mathbf{m}} \mathbf{w}_{\mathbf{m}} = \begin{bmatrix} \dfrac{\partial^2 \mathbf{T}_k}{\partial \widehat{\mathbf{y}}_1 \partial \mathbf{m}} \mathbf{w}_{\mathbf{m}} & \cdots & \dfrac{\partial^2 \mathbf{T}_k}{\partial \widehat{\mathbf{y}}_K \partial \mathbf{m}} \mathbf{w}_{\mathbf{m}} \end{bmatrix}. \tag{5.4}$$

The jth $(s+1)N \times (s+1)N$ block of $\dfrac{\partial^2 \mathbf{T}_k}{\partial \mathbf{y} \partial \mathbf{m}} \mathbf{w}$ is

$$\frac{\partial^2 \mathbf{T}_k}{\partial \widehat{\mathbf{y}}_j \partial \mathbf{m}} \mathbf{w}_{\mathbf{m}} = -\begin{bmatrix} \dfrac{\partial^2 \mathbf{L}\underline{\mathbf{y}}_{k,1}}{\partial \widehat{\mathbf{y}}_j \partial \mathbf{m}} \mathbf{w}_{\mathbf{m}} \\ \vdots \\ \dfrac{\partial^2 \mathbf{L}\underline{\mathbf{y}}_{k,s}}{\partial \widehat{\mathbf{y}}_j \partial \mathbf{m}} \mathbf{w}_{\mathbf{m}} \\ \mathbf{0}_{N \times (s+1)N} \end{bmatrix}.$$

Using the chain rule,

$$\frac{\partial^2 \mathbf{L}\,\underline{\mathbf{y}}_{k,\sigma}}{\partial\,\widehat{\mathbf{y}}_j\,\partial\,\mathbf{m}}\mathbf{w}_\mathbf{m} = \begin{cases} \tau_k \dfrac{\partial^2\left(\mathbf{L}\,\underline{\mathbf{y}}_{k,\sigma}\right)}{\partial\,\underline{\mathbf{y}}_{k,\sigma}\partial\,\mathbf{m}}\mathbf{w}_\mathbf{m}\left(\begin{bmatrix} a_{\sigma,1:s} & 0\end{bmatrix}\otimes\mathbf{I}_N\right) & \text{if } j=k \\ \dfrac{\partial^2\left(\mathbf{L}\,\underline{\mathbf{y}}_{k,\sigma}\right)}{\partial\,\underline{\mathbf{y}}_{k,\sigma}\partial\,\mathbf{m}}\mathbf{w}_\mathbf{m}\left(\begin{bmatrix} \mathbf{0}_{1\times s} & 1\end{bmatrix}\otimes\mathbf{I}_N\right) & \text{if } j=k-1 \\ \mathbf{0}_{N\times(s+1)N} & \text{otherwise.}\end{cases}$$

Letting $\underline{\mathbf{w}}_{k,\sigma} = \mathbf{w}_{k-1} + \tau\sum_{i=1}^{s} a_{\sigma,i}\,\mathbf{W}_{k,i}$, we see that

$$\left(\frac{\partial^2\,\mathbf{T}_k}{\partial\,\mathbf{y}\,\partial\,\mathbf{m}}\mathbf{w}_\mathbf{m}\right)\mathbf{w} = \begin{bmatrix} \left(\dfrac{\partial^2\left(\mathbf{L}\,\underline{\mathbf{y}}_{k,1}\right)}{\partial\,\underline{\mathbf{y}}_{k,1}\partial\,\mathbf{m}}\mathbf{w}_\mathbf{m}\right)\underline{\mathbf{w}}_{k,1} \\ \vdots \\ \left(\dfrac{\partial^2\left(\mathbf{L}\,\underline{\mathbf{y}}_{k,s}\right)}{\partial\,\underline{\mathbf{y}}_{k,s}\partial\,\mathbf{m}}\mathbf{w}_\mathbf{m}\right)\underline{\mathbf{w}}_{k,s} \\ \mathbf{0}_{N\times 1}\end{bmatrix}$$

and

$$\left(\frac{\partial^2\,\mathbf{T}_k}{\partial\,\widehat{\mathbf{y}}_k\,\partial\,\mathbf{m}}\mathbf{w}_\mathbf{m}\right)^\top\widehat{\mathbf{w}}_k = \begin{bmatrix} \tau_k\sum_{\sigma=1}^{s} a_{\sigma,1}\left(\dfrac{\partial^2\left(\mathbf{L}\,\underline{\mathbf{y}}_{k,\sigma}\right)}{\partial\,\underline{\mathbf{y}}_{k,\sigma}\partial\,\mathbf{m}}\mathbf{w}_\mathbf{m}\right)^\top\mathbf{W}_{k,\sigma} \\ \vdots \\ \tau_k\sum_{\sigma=1}^{s} a_{\sigma,s}\left(\dfrac{\partial^2\left(\mathbf{L}\,\underline{\mathbf{y}}_{k,\sigma}\right)}{\partial\,\underline{\mathbf{y}}_{k,\sigma}\partial\,\mathbf{m}}\mathbf{w}_\mathbf{m}\right)^\top\mathbf{W}_{k,\sigma} \\ \mathbf{0}_{N\times 1}\end{bmatrix}$$

$$\left(\frac{\partial^2\,\mathbf{T}_k}{\partial\,\widehat{\mathbf{y}}_{k-1}\,\partial\,\mathbf{m}}\mathbf{w}_\mathbf{m}\right)^\top\widehat{\mathbf{w}}_k = \begin{bmatrix} \mathbf{0}_{sN\times 1} \\ \sum_{\sigma=1}^{s}\left(\dfrac{\partial^2\left(\mathbf{L}\,\underline{\mathbf{y}}_{k,\sigma}\right)}{\partial\,\underline{\mathbf{y}}_{k,\sigma}\partial\,\mathbf{m}}\mathbf{w}_\mathbf{m}\right)^\top\mathbf{W}_{k,\sigma}\end{bmatrix}$$

and $\left(\dfrac{\partial^2\,\mathbf{T}_k}{\partial\,\widehat{\mathbf{y}}_j\,\partial\,\mathbf{m}}\mathbf{w}_\mathbf{m}\right)^\top\widehat{\mathbf{w}}_k = \mathbf{0}_{(s+1)N\times 1}$ for other values of j.

- $\dfrac{\partial\,\widehat{\mathbf{T}}\mathbf{w}}{\partial\,\mathbf{m}}$:

  Finally, we take the derivative of $\widehat{\mathbf{T}}\mathbf{v}$ with respect to $\mathbf{m}$, where $\mathbf{v}$ is some arbitrary vector defined analogously to $\mathbf{y}$ and $\mathbf{w}$. The σ-th stage of the kth time step is

$$\widehat{\mathbf{T}}_{k,\sigma}\mathbf{v} = \begin{cases}\mathbf{v}_k - \mathbf{L}(\mathbf{m})\underline{\mathbf{v}}_{k,\sigma} + \mathbf{V}_{k,\sigma} & \text{if } 1\le\sigma\le s \\ \mathbf{w}_k - \mathbf{w}_{k-1} - \tau_k\sum_{\sigma=1}^{s} b_\sigma\,\mathbf{W}_{k,\sigma} & \text{if } \sigma = s,\end{cases}$$

with $\underline{\mathbf{v}}_{k,\sigma} = \mathbf{v}_{k-1} + \tau \sum_{i=1}^{s} a_{\sigma,i}\, \mathbf{V}_{k,i}$. Taking the derivative of $\widehat{\mathbf{T}}_{k,\sigma}\mathbf{v}$ with respect to $\mathbf{m}$:

$$\frac{\partial\, \widehat{\mathbf{T}}_{k,\sigma}\mathbf{v}}{\partial\, \mathbf{m}} = \begin{cases} -\dfrac{\partial\, \mathbf{L}\underline{\mathbf{v}}_{k,\sigma}}{\partial\, \mathbf{m}} & \text{if } 1 \le \sigma \le s \\ \mathbf{0}_{N\times N_{\mathbf{m}}} & \text{if } \sigma = s. \end{cases}$$

and therefore (cf. (5.2) and (5.3))

$$\frac{\partial\, \widehat{\mathbf{T}}_k\, \mathbf{v}}{\partial\, \mathbf{m}}\, \mathbf{w}_{\mathbf{m}} = \begin{bmatrix} -\dfrac{\partial\, \mathbf{L}\underline{\mathbf{v}}_{k,1}}{\partial\, \mathbf{m}}\mathbf{w}_{\mathbf{m}} \\ \vdots \\ -\dfrac{\partial\, \mathbf{L}\underline{\mathbf{v}}_{k,s}}{\partial\, \mathbf{m}}\mathbf{w}_{\mathbf{m}} \\ \mathbf{0}_{N\times 1} \end{bmatrix} \qquad \text{and} \qquad \frac{\partial\, \widehat{\mathbf{T}}_k\, \mathbf{v}}{\partial\, \mathbf{m}}^{\top} \widehat{\mathbf{w}}_k = -\sum_{\sigma=1}^{s} \frac{\partial\, \mathbf{L}\, \underline{\mathbf{v}}_{k,\sigma}}{\partial\, \mathbf{m}}^{\top} \mathbf{W}_{k,\sigma}.$$

6. Example. The semi-discretized acoustic wave equation on a domain $\Omega \subset \mathbb{R}^d$ in first-order form is

$$\begin{aligned} \dot{p} &= -\boldsymbol{\kappa}\, \nabla \cdot \mathbf{v} + q_p(\mathbf{s}) \\ \dot{v} &= -\boldsymbol{\mu}\, \nabla\, p + q_v(\mathbf{s}), \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} \dot{p} \\ \dot{v} \end{bmatrix} = \underbrace{\begin{bmatrix} & -\boldsymbol{\kappa}\, \nabla\cdot \\ -\boldsymbol{\mu}\, \nabla & \end{bmatrix}}_{\mathbf{L}(\mathbf{m})} \underbrace{\begin{bmatrix} p \\ v \end{bmatrix}}_{\mathbf{y}} + \begin{bmatrix} q_p(\mathbf{s}) \\ q_v(\mathbf{s}) \end{bmatrix}.$$

Here p is pressure, v is particle velocity, $\boldsymbol{\mu}(\mathbf{x}) = 1/\boldsymbol{\rho}(\mathbf{x})$ is the reciprocal of the density of the medium and $\boldsymbol{\kappa}(\mathbf{x})$ is the bulk modulus of the medium. Let $\mathbf{m} = \begin{bmatrix} \mathbf{m}_{\boldsymbol{\kappa}}^{\top} & \mathbf{m}_{\boldsymbol{\mu}}^{\top} \end{bmatrix}^{\top}$ be model parameters so that $\boldsymbol{\kappa} = \boldsymbol{\kappa}\,(\mathbf{m}_{\boldsymbol{\kappa}})$ and $\boldsymbol{\mu} = \boldsymbol{\mu}\,(\mathbf{m}_{\boldsymbol{\mu}})$. The source terms q_p and q_v depend on the parameters $\mathbf{s}$.

To give and explicit example, let us assume we use a finite difference spatial discretization on a staggered grid with N grid points, with p living on the cell nodes (thus having N unknowns) and v on the cell edges (with dN unknowns). Then

$$\mathbf{L}(\mathbf{m}) = -\begin{bmatrix} \mathbf{0} & \operatorname{diag}(\boldsymbol{\kappa})\ \mathbf{D} \\ \operatorname{diag}(\mathbf{A}_n^e \boldsymbol{\mu})\ \mathbf{G} & \mathbf{0} \end{bmatrix}$$

where $\mathbf{D} \in \mathbb{R}^{N\times dN}$ and $\mathbf{G} \in \mathbb{R}^{dN\times N}$ are the discretized versions of the divergence and gradient operators, respectively, and $\mathbf{A}_n^e \in \mathbb{R}^{dN\times N}$ is an averaging operator from the grid nodes to the cell edges. Taking some arbitrary vector $\mathbf{v} = \begin{bmatrix} \mathbf{v}_p^{\top} & \mathbf{v}_v^{\top} \end{bmatrix}^{\top}$ of length $N = N_p + N_v$,

$$-\begin{bmatrix} \mathbf{0} & \operatorname{diag}(\boldsymbol{\kappa})\ \mathbf{D} \\ \operatorname{diag}(\mathbf{A}_n^e \boldsymbol{\mu})\ \mathbf{G} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{v}_p \\ \mathbf{v}_v \end{bmatrix} = -\begin{bmatrix} \operatorname{diag}(\mathbf{D}\, \mathbf{v}_v)\ \boldsymbol{\kappa} \\ \operatorname{diag}(\mathbf{G}\, \mathbf{v}_p)\ \mathbf{A}_n^e \boldsymbol{\mu} \end{bmatrix}$$

and hence

$$\frac{\partial\, (\mathbf{L}(\mathbf{m})\mathbf{v})}{\partial\, \mathbf{m}} = -\begin{bmatrix} \operatorname{diag}(\mathbf{D}\, \mathbf{v}_v)\, \frac{\partial\, \boldsymbol{\kappa}}{\partial\, \mathbf{m}_{\boldsymbol{\kappa}}} & \mathbf{0}_{N_p \times \mathbf{m}_{\boldsymbol{\mu}}} \\ \mathbf{0}_{N_{\mathbf{v}} \times N_{\mathbf{m}_{\boldsymbol{\kappa}}}} & \operatorname{diag}(\mathbf{G}\, \mathbf{v}_p)\, \mathbf{A}_n^e\, \frac{\partial\, \boldsymbol{\mu}}{\partial\, \mathbf{m}_{\boldsymbol{\mu}}} \end{bmatrix}.$$

Letting $\mathbf{w}_{\mathbf{m}} = \begin{bmatrix} \mathbf{w}_{\boldsymbol{\kappa}}^{\top} & \mathbf{w}_{\boldsymbol{\mu}}^{\top} \end{bmatrix}^{\top}$ be an arbitrary vector of length $N_{\mathbf{m}}$ and $\mathbf{w}_y = \begin{bmatrix} \mathbf{w}_p^{\top} & \mathbf{w}_v^{\top} \end{bmatrix}^{\top}$

an arbitrary vector of length $N_p + N_v = (d+1)N$,

$$\frac{\partial\left(\mathbf{L}(\mathbf{m})\mathbf{v}\right)}{\partial\,\mathbf{m}}\,\mathbf{w}_{\mathbf{m}} = -\begin{bmatrix}\operatorname{diag}\left(\mathbf{D}\,\mathbf{v}_v\right)\frac{\partial\,\boldsymbol{\kappa}}{\partial\,\mathbf{m}_{\boldsymbol{\kappa}}}\mathbf{w}_{\boldsymbol{\kappa}}\\ \operatorname{diag}\left(\mathbf{G}\,\mathbf{v}_p\right)\mathbf{A}_n^e\frac{\partial\,\boldsymbol{\mu}}{\partial\,\mathbf{m}_{\boldsymbol{\mu}}}\mathbf{w}_{\boldsymbol{\mu}}\end{bmatrix},$$

$$\frac{\partial\left(\mathbf{L}(\mathbf{m})\mathbf{v}\right)}{\partial\,\mathbf{m}}^{\top}\,\mathbf{w}_y = -\begin{bmatrix}\frac{\partial\,\boldsymbol{\kappa}}{\partial\,\mathbf{m}_{\boldsymbol{\kappa}}}^{\top}\operatorname{diag}\left(\mathbf{D}\,\mathbf{v}_v\right)\mathbf{w}_p\\ \frac{\partial\,\boldsymbol{\mu}}{\partial\,\mathbf{m}_{\boldsymbol{\mu}}}^{\top}\mathbf{A}_n^{e\top}\operatorname{diag}\left(\mathbf{G}\,\mathbf{v}_p\right)\mathbf{w}_v\end{bmatrix},$$

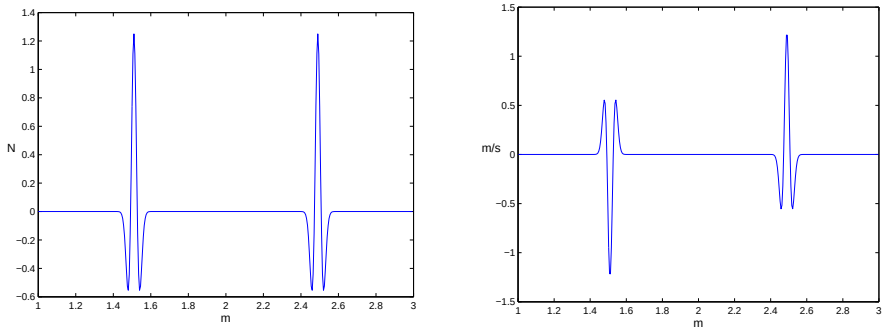
$$\frac{\partial^2\left(\mathbf{L}(\mathbf{m})\mathbf{v}\right)}{\partial\,\mathbf{m}\,\partial\,\mathbf{m}}\,\mathbf{w}_{\mathbf{m}} = -\begin{bmatrix}\operatorname{diag}\left(\mathbf{D}\,\mathbf{v}_v\right)\frac{\partial^2\,\boldsymbol{\kappa}}{\partial\,\mathbf{m}_{\boldsymbol{\kappa}}\,\partial\,\mathbf{m}_{\boldsymbol{\kappa}}}\mathbf{w}_{\boldsymbol{\kappa}} & \mathbf{0}_{N\times N_{\mathbf{m}_{\boldsymbol{\mu}}}}\\ \mathbf{0}_{dN\times N_{\mathbf{m}_{\boldsymbol{\kappa}}}} & \operatorname{diag}\left(\mathbf{G}\,\mathbf{v}_p\right)\mathbf{A}_n^e\frac{\partial^2\,\boldsymbol{\mu}}{\partial\,\mathbf{m}_{\boldsymbol{\mu}}\,\partial\,\mathbf{m}_{\boldsymbol{\mu}}}\mathbf{w}_{\boldsymbol{\mu}}\end{bmatrix},$$

$$\frac{\partial^2\left(\mathbf{L}(\mathbf{m})\mathbf{v}\right)}{\partial\,\mathbf{v}\,\partial\,\mathbf{m}}\,\mathbf{w}_{\mathbf{m}} = -\begin{bmatrix}\mathbf{0}_{N\times N} & \operatorname{diag}\left(\frac{\partial\,\boldsymbol{\kappa}}{\partial\,\mathbf{m}_{\boldsymbol{\kappa}}}\mathbf{w}_{\boldsymbol{\kappa}}\right)\mathbf{D}\\ \operatorname{diag}\left(\mathbf{A}_n^e\frac{\partial\,\boldsymbol{\mu}}{\partial\,\mathbf{m}_{\boldsymbol{\mu}}}\mathbf{w}_{\boldsymbol{\mu}}\right)\mathbf{G} & \mathbf{0}_{dN\times dN}\end{bmatrix}$$

The first two products are needed when computing the action of the sensitivity matrix and its transpose, and the last two are additionally needed when computing the action of the full Hessian. Notice that $\dfrac{\partial^2\left(\mathbf{L}(\mathbf{m})\mathbf{v}\right)}{\partial\,\mathbf{v}\,\partial\,\mathbf{m}}\,\mathbf{w}_{\mathbf{m}}$ is just $\mathbf{L}$ once we replace $\boldsymbol{\kappa}$ and $\boldsymbol{\mu}$ by $\frac{\partial\,\boldsymbol{\kappa}}{\partial\,\mathbf{m}_{\boldsymbol{\kappa}}}\mathbf{w}_{\boldsymbol{\kappa}}$ and $\frac{\partial\,\boldsymbol{\mu}}{\partial\,\mathbf{m}_{\boldsymbol{\mu}}}\mathbf{w}_{\boldsymbol{\mu}}$ respectively.

We now consider a simple toy problem to illustrate that the order of accuracy of the adjoint RK method is the same of the corresponding forward RK method, and the gradient computations of the misfit function therefore retain the same order of accuracy. An investigation of the effect that higher-order time-stepping schemes have on the quality of the recovered parameters is beyond the scope of this paper and will be considered at a later point for a more realistic problem than the one presented here.

The test case we consider is the one-dimensional acoustic wave equation on a domain $\Omega = [0, 5]$m with periodic boundary conditions. The grid is spatially staggered, with the pressure variables located at the nodes and the velocity terms located at the interval midpoints. To minimize the effect of numerical dispersion we partition the domain into 1000 equally-sized intervals and use 8th-order spatial derivative operators.



(a) Pressure

(b) Velocity

Fig. 1: Initial condition $\mathbf{y}_0$

The integration is performed from $0s$ to $4s$ and the initial condition is given by the pres-

sure p and velocity v shown in Figure 1; when starting from this initial condition the left pressure wavelet is moving to the left and the right wavelet is moving to the right. There is no active source term for the forward problem.

For us to focus solely on the order of accuracy, we use the following misfit functional

$$\mathcal{M} = \frac{1}{2}\left\|\mathbf{d} - \mathbf{d}^{\text{obs}}\right\|^2 = \frac{1}{2}\left\|\mathbf{Q}\,\mathbf{y} - \mathbf{d}^{\text{obs}}\right\|^2.$$

Here $\mathbf{Q} = \text{diag}(\mathbf{Q}_k)$ and $\mathbf{Q}_k$ are projection operators that simply extract the solution at

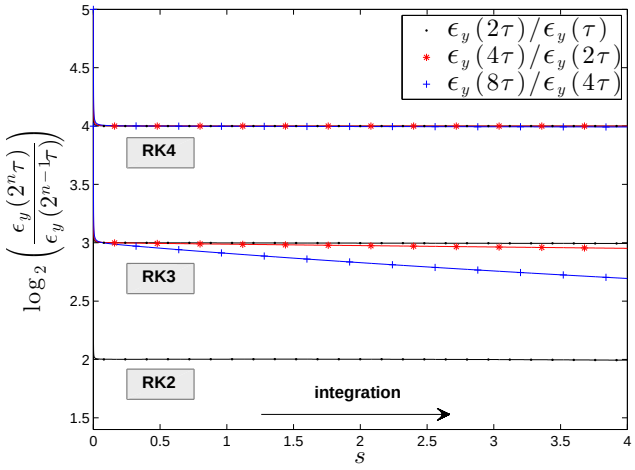
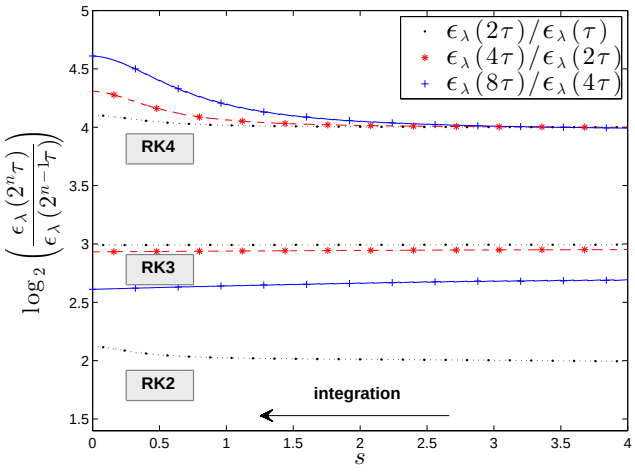


Fig. 2: Order of accuracy of the forward solution with $\tau = 10^{-4}s$.

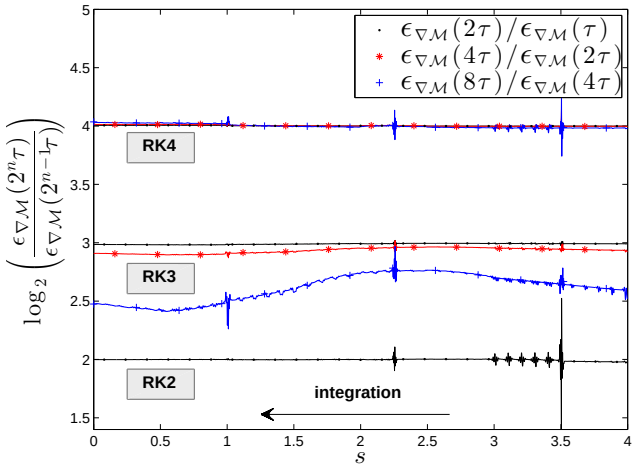
certain observation times corresponding to a subset of the time steps $1, \ldots, K$ and we assume a complete and noise-free observation of the solution at those time steps; therefore $\mathbf{Q}_k$ is the identity matrix at these time steps and the zero matrix at all other time steps. The observed data $\mathbf{d}^{\text{obs}}$ consists simply of the initial condition $\mathbf{y}_0$ at these observation times, so this misfit is unrealistic but it allows us to concentrate on the task at hand. It's clear that in this case $\nabla_{\mathbf{d}}\mathcal{M} = \mathbf{d} - \mathbf{d}^{\text{obs}}$ (the data residual) and $\frac{\partial \mathbf{d}}{\partial \mathbf{y}} = \mathbf{Q}$. The data residual forms the adjoint source term for this problem and is simply $\mathbf{y}_k - \mathbf{y}_0$ at observation times and $\mathbf{0}$ otherwise.

We let the bulk modulus be $\kappa = 2\text{N}$ and the density be $\rho = \frac{1}{2}\frac{\text{kg}}{\text{m}}$ uniformly on Ω, resulting in a wave velocity of $2\frac{\text{m}}{\text{s}}$, and solve the forward problem to high precision using RK4 with a very small time step $\tau_{\text{fine}} = 10^{-6}s$, which we subsequently take to be the true solution $\mathbf{y}^{\text{true}}$. We then compute the corresponding adjoint solution $\boldsymbol{\lambda}^{\text{true}}$ and gradient $\nabla\mathcal{M}^{\text{true}}$ using Algorithm 3.2. Observations are taken every $0.1s$.

The forward solutions are then computed using explicit RK methods of orders 2, 3 and 4 and time steps $\tau = 10^{-5}s$, 2τ, 4τ and 8τ, which we denote by $\mathbf{y}^{\text{RKs}}(2^n\tau)$, where $s = 2, 3, 4$ and $n = 0, 1, 2, 3$. The corresponding adjoint solutions $\boldsymbol{\lambda}^{\text{RKs}}(2^n\tau)$ and misfit gradients $\nabla\mathcal{M}^{\text{RKs}}(2^n\tau)$ are then computed with Algorithm 3.2. To determine the order of accuracy, the absolute errors $\boldsymbol{\epsilon}^s_{\mathbf{y},k}(2^n\tau) = \|\mathbf{y}^s_k(2^n\tau) - \mathbf{y}^{\text{true}}_k\|$, $\boldsymbol{\epsilon}^s_{\boldsymbol{\lambda},k}(2^n\tau) = \|\boldsymbol{\lambda}^s_k(2^n\tau) - \boldsymbol{\lambda}^{\text{true}}_k\|$ and $\boldsymbol{\epsilon}^s_{\nabla\mathcal{M},k}(2^n\tau) = \|\nabla\mathcal{M}^s_k(2^n\tau) - \nabla\mathcal{M}^{\text{true}}_k\|$ are computed at each time step k that is common to all of the above time-stepping schemes. The binary logarithms of the ratios $\boldsymbol{\epsilon}^s_{\mathbf{y},k}(2^n\tau)/\boldsymbol{\epsilon}^s_{\mathbf{y},k}(2^{n-1}\tau)$ are plotted in Figure 2 and the binary logarithms of the ratios $\boldsymbol{\epsilon}^s_{\boldsymbol{\lambda},k}(2^n\tau)/\boldsymbol{\epsilon}^s_{\boldsymbol{\lambda},k}(2^{n-1}\tau)$ and $\boldsymbol{\epsilon}^s_{\nabla\mathcal{M},k}(2^n\tau)/\boldsymbol{\epsilon}^{\text{RKs}}_{\nabla\mathcal{M},k}(2^{n-1}\tau)$ are shown in Figure 3. The second-order RK method is unstable for time steps larger than 2τ, so only the ratio $\boldsymbol{\epsilon}^2_{\nabla\mathcal{M}}(2\tau)/\boldsymbol{\epsilon}^2_{\nabla\mathcal{M}}(\tau)$ is shown.



(a)



(b)

Fig. 3: Order of accuracy of the adjoint solution (top), the misfit gradient (bottom) for RK methods, with $\tau = 10^{-4}s$. The adjoint solution is computed with observations every $0.1s$.

Plotting the binary logarithms of the error ratios means that we expect to see values of around 4 for fourth-order schemes, around 3 for third-order schemes and around 2 for second-order schemes. This is indeed the case, with the forward solution clearly exhibiting this expected behaviour in Figure 2, with the exception of the ratio $\boldsymbol{\epsilon}^3_{\nabla\mathcal{M}}(8\tau)/\boldsymbol{\epsilon}^3_{\nabla\mathcal{M}}(4\tau)$ showing some decrease in accuracy due to the time step 8τ being slightly too large for the third-order RK method.

Figure 3 shows that this behaviour translates to the adjoint and gradient computations, with the order of accuracy of the adjoint solution and gradient computation matching up with that of its corresponding forward solution, as was to be expected from our discussion in Section 4.3. There are some slight (and unexplained) deviations that can be observed for

the adjoint RK4 method where the order of accuracy actually seems to increase somewhat as the integration moves along, forming a hump in the ratios. Deviations such as these are also exhibited when using observation times other than every $0.1s$, although they are usually minor and the order of accuracy returns to its expected value after a certain amount of time. Interestingly, deviations in the order of accuracy of the adjoint solution rarely affect the order of accuracy of the misfit gradient, an example of which can be seen in Figure 3b, where the hump seen for the adjoint RK4 method does not affect the misfit gradient at all. The misfit gradient instead exhibits its own short-term, highly oscillatory deviations that peak around the observation times, the times at which data residuals are included in the adjoint source term.

The exception to this is the RK3 method with larger time steps, where the order of accuracy is significantly less than 3 from the start. In fact this must be the case since the data residual in the adjoint source term at $4s$ will have the inaccurate forward solution at this time in it, so the inaccuracy that was present in the forward solution necessarily means that the adjoint solution will also be inaccurate from the start of the integration. In addition, the adjoint RK3 method itself is inaccurate because of the larger time step, although interestingly the loss of accuracy is not quite as pronounced as that for the forward method. The inaccuracies in both the forward and adjoint solutions will of course also affect the accuracy of the misfit gradient, as is clear in Figure 3b. All these deviations disappear when using smaller time steps.

We have restricted the discussion to RK methods because, as we have seen, adjoint LM methods are essentially the same as their corresponding forward methods, so a numerical investigation will not reveal anything of interest.

7. Final Remarks and Future Work. We have shown that the application of the discrete adjoint method to optimization problems constrained by linear time-dependent PDEs leads to adjoint LM time-stepping schemes that have the same form as their corresponding forward schemes, ignoring the final few steps of the adjoint scheme that correspond to the initialization steps of the forward method. Adjoint RK methods look different than their forward counterparts, but in Section 4.3 we gave a simple argument to show that they have the same order of accuracy.

A crucial point of difference between the forward and adjoint methods is in the handling of the source terms. In the case of Adams-type methods the source term at a given time step is a linear combination of the source function at previous (and possibly the current) time steps, whereas for BDF the source term is a rescaling of the source function at the current time. In the case of adjoint methods this is not the case, one simply takes the value of the adjoint source function at the current time step. For forward RK methods the source term enters in the internal stages, whereas in adjoint RK methods we will actually have the source term appearing only in the update for $\boldsymbol{\lambda}_k$ and not in the internal stages at all. The value of the adjoint source function is hence needed only at the time t_k.

We have allowed for variable time steps in the RK method, the size of which would usually be determined using some embedded RK method to provide a step size that ensures the solution satisfies a given accuracy. These step sizes are inherited by the adjoint RK methods and some readers might question whether this might not lead to a loss of accuracy in the adjoint method due to some time steps possibly being too large. However, while accuracy of the adjoint solution might be a concern when working in the OD framework, we are instead interested in solving the adjoint problem corresponding to the fully-discretized forward problem. In the DO framework it is therefore not necessarily clear what solving the adjoint problem "to a given accuracy" would mean. This does not apply to the order of accuracy of course, it is desirable that the time-discretization error in the sentivity matrix computations

decreases at the same rate as that of the forward problem.

The internal stages of the RK method need to be stored since the derivatives of $\mathbf{T}$ have to have access to them, leading to an increase in storage overhead; for explicit RK methods the last internal stage does not need to be stored since it will not be used. LM methods have the advantage that they do not have internal stages that need to be stored, but explicit higher-order RK methods allow for larger time steps than Adams methods so they may still need less required storage space overall in some situations.

Although the work in this paper is restricted to linear PDEs, it is applicable to several important problems, including the acoustic and elastic wave equations, Maxwell's equations and the advection-diffusion equation. For nonlinear problems solved with LM and RK schemes the corresponding adjoint schemes will be the same as the ones found in Algorithms 4.1 and 4.2, except that $\mathbf{L}$ is replaced by the linearization of the PDE with respect to the forward solution at the current time step.

The strength of the technique that we have used to study the discrete adjoint method for linear time-dependent PDEs in this paper is that it can easily be extended to nonlinear PDEs that are solved with more interesting integration methods, such as IMEX [32], staggered [13] or exponential integration schemes [21]. These will be the subject of future work, and more than anything the work done in this article, while interesting in its own right, is meant to provide a foundation on which we can build to find the adjoint time-stepping methods for these schemes.

Appendix A. Computing the action of the Hessian $\mathcal{H}\mathcal{M}$.

The adjoint method can also be used to find the expression for the Hessian multiplied by some arbitrary vector $\mathbf{w} = \begin{bmatrix}\mathbf{w_m}^\top & \mathbf{w_s}^\top\end{bmatrix}^\top$ of length $N_\mathbf{m} + N_\mathbf{s}$. Abusing notation, we let $\frac{\partial^2 \mathbf{V}}{\partial \mathbf{x}\,\partial \mathbf{y}}\mathbf{z} = \frac{\partial}{\partial \mathbf{x}}\left(\frac{\partial \mathbf{V}}{\partial \mathbf{y}}\mathbf{z}\right)$, where $\mathbf{V}, \mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ are placeholders for various terms that show up in what follows. We know that

$$\nabla_\mathbf{p}\mathcal{M} = \frac{\partial\,\mathbf{d}}{\partial\,\mathbf{p}}^\top \nabla_\mathbf{d}\mathcal{M} = \mathbf{J}^\top \nabla_\mathbf{d}\mathcal{M},$$

so by the product rule

$$\begin{aligned}\mathcal{H}_\mathcal{M}\,\mathbf{w} = \frac{\partial^2 \mathcal{M}}{\partial\,\mathbf{p}\,\partial\,\mathbf{p}}\,\mathbf{w} = \nabla_\mathbf{p}\left(\nabla_\mathbf{p}\mathcal{M}^\top\,\mathbf{w}\right) = \nabla_\mathbf{p}\left(\nabla_\mathbf{d}\mathcal{M}^\top\,\mathbf{J}\,\mathbf{w}\right)& \\ = \underset{①}{\frac{\partial^2 \mathcal{M}}{\partial\,\mathbf{p}\,\partial\,\mathbf{d}}^\top}\,\mathbf{J}\,\mathbf{w} + \underset{②}{\frac{\partial\left(\mathbf{J}\,\mathbf{w}\right)}{\partial\,\mathbf{p}}^\top}\nabla_\mathbf{d}\mathcal{M}\,.&\end{aligned}$$

By the chain rule, ① is just

$$\frac{\partial^2 \mathcal{M}}{\partial\,\mathbf{p}\,\partial\,\mathbf{d}}^\top \mathbf{J}\,\mathbf{w} = \frac{\partial\,\mathbf{d}}{\partial\,\mathbf{p}}^\top \frac{\partial^2 \mathcal{M}}{\partial\,\mathbf{d}\,\partial\,\mathbf{d}}\,\mathbf{J}\,\mathbf{w} = \mathbf{J}^\top \frac{\partial^2 \mathcal{M}}{\partial\,\mathbf{d}\,\partial\,\mathbf{d}}\,\mathbf{J}\,\mathbf{w}.$$

Since $\mathbf{J}\,\mathbf{w} = \frac{\partial\,\mathbf{d}}{\partial\,\mathbf{p}}\,\mathbf{w} = \frac{\partial \mathbf{d}}{\partial \mathbf{y}}\mathbf{x}$ with $\mathbf{x} = \mathbf{x}(\mathbf{p}) = \frac{\partial \mathbf{y}}{\partial\,\mathbf{p}}\,\mathbf{w}$, ② becomes

$$\begin{aligned}\left(\frac{\partial}{\partial\,\mathbf{p}}\mathbf{J}\,\mathbf{w}\right)^{\top}\nabla_{\mathbf{d}}\mathcal{M} &= \left(\frac{\partial}{\partial\,\mathbf{p}}\left(\frac{\partial \mathbf{d}}{\partial \mathbf{y}}\mathbf{x}\right)\right)^{\top}\nabla_{\mathbf{d}}\mathcal{M}\\ &= \frac{\partial \mathbf{y}}{\partial\,\mathbf{p}}^{\top}\left(\frac{\partial}{\partial\,\mathbf{y}}\left(\frac{\partial \mathbf{d}}{\partial\,\mathbf{y}}\,\mathbf{x}\right)\Big|_{\mathbf{x}}\right)^{\top}\nabla_{\mathbf{d}}\mathcal{M} + \frac{\partial\,\mathbf{x}}{\partial\,\mathbf{p}}^{\top}\frac{\partial\,\mathbf{d}}{\partial\,\mathbf{y}}^{\top}\nabla_{\mathbf{d}}\mathcal{M}\\ &= \frac{\partial \mathbf{y}}{\partial\,\mathbf{p}}^{\top}\underbrace{\left(\frac{\partial^2\,\mathbf{d}}{\partial\,\mathbf{y}\,\partial\,\mathbf{y}}\,\mathbf{x}\right)^{\top}}_{③}\nabla_{\mathbf{d}}\mathcal{M} + \frac{\partial\,\mathbf{x}}{\partial\,\mathbf{p}}^{\top}\underbrace{\frac{\partial\,\mathbf{d}}{\partial\,\mathbf{y}}^{\top}}_{④}\nabla_{\mathbf{d}}\mathcal{M}\,.\end{aligned}$$

Consider

$$④ = \frac{\partial\,\mathbf{x}}{\partial\,\mathbf{p}}^{\top}\frac{\partial\,\mathbf{d}}{\partial\,\mathbf{y}}^{\top}\nabla_{\mathbf{d}}\mathcal{M} = \left(\frac{\partial^2\,\mathbf{y}}{\partial\,\mathbf{p}\,\partial\,\mathbf{p}}\mathbf{w}\right)^{\top}\frac{\partial\,\mathbf{d}}{\partial\,\mathbf{y}}^{\top}\nabla_{\mathbf{d}}\mathcal{M}.$$

Each term of

$$\begin{aligned}\frac{\partial^2\,\mathbf{y}}{\partial\,\mathbf{p}\,\partial\,\mathbf{p}}\mathbf{w} &= \left[\frac{\partial}{\partial\,\mathbf{m}}\left(\frac{\partial \mathbf{y}}{\partial\,\mathbf{m}}\,\mathbf{w}_{\mathbf{m}} + \frac{\partial \mathbf{y}}{\partial\,\mathbf{s}}\,\mathbf{w}_{\mathbf{s}}\right)\quad\frac{\partial}{\partial\,\mathbf{s}}\left(\frac{\partial \mathbf{y}}{\partial\,\mathbf{m}}\,\mathbf{w}_{\mathbf{m}} + \frac{\partial \mathbf{y}}{\partial\,\mathbf{s}}\,\mathbf{w}_{\mathbf{s}}\right)\right]\\ &= \begin{bmatrix}\underbrace{\frac{\partial^2\,\mathbf{y}}{\partial\,\mathbf{m}\,\partial\,\mathbf{m}}\mathbf{w}_{\mathbf{m}}}_{⑤} + \underbrace{\frac{\partial^2\,\mathbf{y}}{\partial\,\mathbf{m}\,\partial\,\mathbf{s}}\mathbf{w}_{\mathbf{s}}}_{⑥} & \underbrace{\frac{\partial^2\,\mathbf{y}}{\partial\,\mathbf{s}\,\partial\,\mathbf{m}}\mathbf{w}_{\mathbf{m}}}_{⑦} + \underbrace{\frac{\partial^2\,\mathbf{y}}{\partial\,\mathbf{s}\,\partial\,\mathbf{s}}\mathbf{w}_{\mathbf{s}}}_{⑧}\end{bmatrix}\end{aligned}$$

will be considered in turn. For ⑤ and ⑦, start with $\frac{\partial\,\mathbf{T}(\mathbf{m})}{\partial\,\mathbf{m}} + \widehat{\mathbf{T}}\,\frac{\partial \mathbf{y}}{\partial\,\mathbf{m}} = \mathbf{0}$, multiply by $\mathbf{w}_{\mathbf{m}}$ and set $\mathbf{z} = \frac{\partial\,\mathbf{T}(\mathbf{m})}{\partial\,\mathbf{m}}\mathbf{w}_{\mathbf{m}}$ and $\mathbf{v}_{\mathbf{m}} = \frac{\partial \mathbf{y}}{\partial\,\mathbf{m}}\mathbf{w}_{\mathbf{m}}$ for compactness. Then:

- ⑤: Differentiating $\mathbf{z} + \widehat{\mathbf{T}}\,\mathbf{v}_{\mathbf{m}} = \mathbf{0}$ with respect to $\mathbf{m}$:

$$\frac{\partial\,\mathbf{z}}{\partial\,\mathbf{m}}\Big|_{\mathbf{y}} + \frac{\partial\,\mathbf{z}}{\partial\,\mathbf{y}}\,\frac{\partial \mathbf{y}}{\partial\,\mathbf{m}} + \frac{\partial(\widehat{\mathbf{T}}\mathbf{v}_{\mathbf{m}})}{\partial\,\mathbf{m}}\Big|_{\mathbf{v}_{\mathbf{m}}} + \widehat{\mathbf{T}}\,\frac{\partial\,\mathbf{v}_{\mathbf{m}}}{\partial\,\mathbf{m}} = \mathbf{0},$$

  and therefore

$$⑤ = \frac{\partial^2\,\mathbf{y}}{\partial\,\mathbf{m}\,\partial\,\mathbf{m}}\mathbf{w}_{\mathbf{m}} = \frac{\partial\,\mathbf{v}_{\mathbf{m}}}{\partial\,\mathbf{m}} = -\widehat{\mathbf{T}}^{-1}\left(\frac{\partial\,\mathbf{z}}{\partial\,\mathbf{m}}\Big|_{\mathbf{y}} + \frac{\partial\,\mathbf{z}}{\partial\,\mathbf{y}}\,\frac{\partial \mathbf{y}}{\partial\,\mathbf{m}} + \frac{\partial(\widehat{\mathbf{T}}\mathbf{v}_{\mathbf{m}})}{\partial\,\mathbf{m}}\Big|_{\mathbf{v}_{\mathbf{m}}}\right).$$

- ⑦: Similarly, differentiating $\mathbf{z} + \widehat{\mathbf{T}}\,\mathbf{v}_{\mathbf{m}} = \mathbf{0}$ with respect to $\mathbf{s}$ gives

$$⑦ = \frac{\partial^2\,\mathbf{y}}{\partial\,\mathbf{s}\,\partial\,\mathbf{m}}\mathbf{w}_{\mathbf{m}} = -\widehat{\mathbf{T}}^{-1}\,\frac{\partial\,\mathbf{z}}{\partial\,\mathbf{y}}\,\frac{\partial \mathbf{y}}{\partial\,\mathbf{s}}.$$

For ⑥ and ⑧, start with $\widehat{\mathbf{T}}\,\frac{\partial \mathbf{y}}{\partial\,\mathbf{s}} = \frac{\partial \mathbf{q}}{\partial\,\mathbf{s}}$, multiply by $\mathbf{w}_{\mathbf{s}}$ and set $\mathbf{v}_{\mathbf{s}} = \frac{\partial \mathbf{y}}{\partial\,\mathbf{s}}\mathbf{w}_{\mathbf{s}}$ for compactness. Then:

- ⑧: Differentiate $\widehat{\mathbf{T}}\,\mathbf{v}_{\mathbf{s}} = \frac{\partial \mathbf{q}}{\partial\,\mathbf{s}}$ with respect to $\mathbf{s}$:

$$⑧ = \frac{\partial^2\,\mathbf{y}}{\partial\,\mathbf{s}\,\partial\,\mathbf{s}}\mathbf{w}_{\mathbf{s}} = \widehat{\mathbf{T}}^{-1}\frac{\partial^2\,\mathbf{q}}{\partial\,\mathbf{s}\,\partial\,\mathbf{s}}\mathbf{w}_{\mathbf{s}}.$$

- ⑥: Similarly, differentiate $\widehat{\mathbf{T}}\,\mathbf{v_s} = \frac{\partial \mathbf{q}}{\partial \mathbf{s}}$ with respect to $\mathbf{m}$:

$$⑥ = \frac{\partial^2 \mathbf{y}}{\partial \mathbf{m}\,\partial \mathbf{s}}\mathbf{w_s} = -\widehat{\mathbf{T}}^{-1}\ \left.\frac{\partial(\widehat{\mathbf{T}}\mathbf{v_s})}{\partial \mathbf{m}}\right|_{\mathbf{v_s}}.$$

Plugging all of this into ④ and using the fact that $\boldsymbol{\lambda} = \widehat{\mathbf{T}}^{-\top}\,\frac{\partial \mathbf{d}}{\partial \mathbf{y}}^{\top}\,\nabla_{\mathbf{d}}\mathcal{M}$:

$$\begin{aligned}
④ &= -\begin{bmatrix} \left(\left.\frac{\partial \mathbf{z}}{\partial \mathbf{m}}\right|_{\mathbf{y}}\right)^{\top} + \frac{\partial \mathbf{y}}{\partial \mathbf{m}}^{\top}\frac{\partial \mathbf{z}}{\partial \mathbf{y}}^{\top} + \left(\left.\frac{\partial(\widehat{\mathbf{T}}\mathbf{v_m})}{\partial \mathbf{m}}\right|_{\mathbf{v_m}}\right)^{\top} \\ \frac{\partial \mathbf{y}}{\partial \mathbf{s}}^{\top}\frac{\partial \mathbf{z}}{\partial \mathbf{y}}^{\top} - \left(\frac{\partial^2 \mathbf{q}}{\partial \mathbf{s}\,\partial \mathbf{s}}\mathbf{w_s}\right)^{\top} \end{bmatrix}\boldsymbol{\lambda} \\
&= -\frac{\partial \mathbf{y}}{\partial \mathbf{p}}^{\top}\frac{\partial \mathbf{z}}{\partial \mathbf{y}}^{\top}\boldsymbol{\lambda} - \begin{bmatrix} \left(\left.\frac{\partial \mathbf{z}}{\partial \mathbf{m}}\right|_{\mathbf{y}}\right)^{\top} + \left(\left.\frac{\partial(\widehat{\mathbf{T}}\mathbf{v_m})}{\partial \mathbf{m}}\right|_{\mathbf{v_m}}\right)^{\top} \\ -\left(\frac{\partial^2 \mathbf{q}}{\partial \mathbf{s}\,\partial \mathbf{s}}\mathbf{w_s}\right)^{\top} \end{bmatrix}\boldsymbol{\lambda}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\frac{\partial^2 \mathcal{M}}{\partial \mathbf{p}\,\partial \mathbf{p}}\,\mathbf{w} &= ① + ② = ① + ③ + ④ \\
&= \mathbf{J}^{\top}\frac{\partial^2 \mathcal{M}}{\partial \mathbf{d}\,\partial \mathbf{d}}\,\mathbf{J}\,\mathbf{w} + \frac{\partial \mathbf{y}}{\partial \mathbf{p}}^{\top}\left(\frac{\partial^2 \mathbf{d}}{\partial \mathbf{y}\,\partial \mathbf{y}}\mathbf{v}\right)^{\top}\nabla_{\mathbf{d}}\mathcal{M} - \frac{\partial \mathbf{y}}{\partial \mathbf{p}}^{\top}\frac{\partial \mathbf{z}}{\partial \mathbf{y}}^{\top}\boldsymbol{\lambda} + \\
&\qquad - \begin{bmatrix} \left(\left.\frac{\partial \mathbf{z}}{\partial \mathbf{m}}\right|_{\mathbf{y}}\right)^{\top} + \frac{\partial \widehat{\mathbf{T}}\mathbf{v}}{\partial \mathbf{m}}^{\top} \\ -\left(\frac{\partial^2 \mathbf{q}}{\partial \mathbf{s}\,\partial \mathbf{s}}\mathbf{w_s}\right)^{\top} \end{bmatrix}\boldsymbol{\lambda} \\
&= \begin{bmatrix} -\frac{\partial \mathbf{T}}{\partial \mathbf{m}}^{\top} \\ \frac{\partial \mathbf{q}}{\partial \mathbf{s}}^{\top} \end{bmatrix}\boldsymbol{\mu} - \begin{bmatrix} \left(\frac{\partial^2 \mathbf{T}}{\partial \mathbf{m}\,\partial \mathbf{m}}\mathbf{w_m}\right)^{\top} + \frac{\partial \widehat{\mathbf{T}}\mathbf{v}}{\partial \mathbf{m}}^{\top} \\ -\left(\frac{\partial^2 \mathbf{q}}{\partial \mathbf{s}\,\partial \mathbf{s}}\mathbf{w_s}\right)^{\top} \end{bmatrix}\boldsymbol{\lambda}
\end{aligned}$$

with $\boldsymbol{\mu} = \widehat{\mathbf{T}}^{-\top}\left(\frac{\partial \mathbf{d}}{\partial \mathbf{y}}^{\top}\frac{\partial^2 \mathcal{M}}{\partial \mathbf{d}\,\partial \mathbf{d}}\frac{\partial \mathbf{d}}{\partial \mathbf{y}}\mathbf{v} + \left(\frac{\partial^2 \mathbf{d}}{\partial \mathbf{y}\,\partial \mathbf{y}}\mathbf{v}\right)^{\top}\nabla_{\mathbf{d}}\mathcal{M} - \left(\frac{\partial^2 \mathbf{T}}{\partial \mathbf{y}\,\partial \mathbf{m}}\mathbf{w_m}\right)^{\top}\boldsymbol{\lambda}\right)$.

REFERENCES

[1] T. APEL AND T. G. FLAIG, *Crank-Nicolson schemes for optimal control problems with evolution equations*, SIAM Journal on Numerical Analysis, 50 (2012), pp. 1484–1512.

[2] U. ASCHER AND E. HABER, *Preconditioned all-at-once methods for large, sparse parameter-estimation problems*, Inverse Problems, 17 (2001), pp. 1847–1864.

[3] U. ASCHER AND L. PETZOLD, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, SIAM, 1998.

[4] R. C. ASTER, B. BORCHERS, AND C. H. THURBER, *Parameter Estimation and Inverse Problems*, Academic Press, 2 ed., 2012.

[5] H. G. BOCK, T. CARRARO, W. JÄGER, S. KÖRKEL, R. RANNACHER, AND J. SCHLÖDER, eds., *Model Based Parameter Estimation: Theory and Applications*, Springer, 2013.

[6] P. BRENNER, M. CROUZEIX, AND V. THOMÉE, *Single step methods for inhomogeneous linear differential equations in Banach space*, RAIRO Analyse Numerique, 16 (1982), pp. 5–26.

[7] Y. CAO, S. LI, L. PETZOLD, AND R. SERBAN, *Adjoint sensitivity analysis for differential-algebraic equations: the adjoint DAE system and its numerical solution*, SIAM J. Scient. Comput., 24 (2003), pp. 1076–1089.

[8] I.-C. CHOU AND E. O. VOIT, *Recent developments in parameter estimation and structure identification of biochemical and genomic systems*, Mathematical Biosciences, 219 (2009), pp. 57–83.

[9] J. E. DENNIS AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM, 1996.

[10] H. W. ENGL, M. HANKE, AND A. NEUBAUER, *Regularization of Inverse Problems*, Kluwer, 1996.

[11] A. FICHTNER, *Full Seismic Waveform Modeling and Inversion*, Springer, 2011.

[12] C. W. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, 1971.

[13] M. GHRIST, B. FORNBERG, AND T. A. DRISCOLL, *Staggered time integrators for wave equations*, SIAM Journal of Numerical Analysis, 38 (2000), pp. 718–741.

[14] R. GRIESSE AND A. WALTHER, *Evaluating Gradients in Optimal Control: Continuous Adjoints versus Automatic Differentiation*, Journal of Optimization Theory and Applications, 122 (2004), pp. 63–86.

[15] M. D. GUNZBURGER, *Perspectives in Flow Control and Optimization*, SIAM, 2002.

[16] E. HABER, *Computational Methods in Geophysical Electromagnetics*, SIAM, 2014.

[17] W. HAGER, *Runge-Kutta methods in optimal control and the transformed adjoint system*, Numerische Mathematik, 87 (2000), pp. 247–282.

[18] E. HAIRER, S. P. NØRSETT, AND G. WANNER, *Solving Ordinary Differential Equations I: Nonstiff Problems*, Springer, 2 ed., 1993.

[19] E. HAIRER AND G. WANNER, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, Springer, 2 ed., 1996.

[20] M. HEINKENSCHLOSS, *Numerical solution of implicitly constrained optimization problems*, Tech. Report TR08-05, Rice University, Department of Computational and Applied Mathermatics, 2013.

[21] M. HOCHBRUCK AND A. OSTERMANN, *Exponential integrators*, Acta Numerica, 19 (1986), pp. 209–286.

[22] V. ISAKOV, *Inverse Problems for Partial Differential Equations*, Springer; 2nd edition, 2006.

[23] J. B. KOOL, J. C. PARKER, AND M. T. VAN GENUCHTEN, *Parameter estimation for unsaturated flow and transport models: A review*, Journal of Hydrology, 91 (1986), pp. 255–293.

[24] D. MEIDNER AND B. VEXLER, *A priori error analysis of the Petrov-Galerkin Crank-Nicolson scheme for Parabolic Optimal Control Problems*, SIAM Journal on Control and Optimization, 49 (2011), pp. 2183–2211.

[25] S. K. NADARAJAH AND A. JAMESON, *A Comparison of the Continuous and Discrete Adjoint Approach to Automatic Aerodynamic Optimization*, 38th Aerospace Sciences Meeting and Exhibit, January 2000, Reno, NV, (2000).

[26] S. K. NADARAJAH, A. JAMESON, AND J. ALONSO, *An Adjoint Method for the Calculation of Remote Sensitivities in Supersonic Flow*, 40th Aerospace Sciences Meeting and Exhibit, January 2002, Reno, NV, (2002).

[27] J. NOCEDAL AND S. J. WRIGHT, *Numerical Optimization*, Springer, 2 ed., 2006.

[28] A. OSTERMANN AND M. ROCHE, *Runge-Kutta methods for partial differential equations and fractional orders of convergence*, Mathematics of Computing, 59 (1992), pp. 403–420.

[29] D. I. PAPADIMITRIOU AND K. C. GIANNAKOGLOU, *Computation of the Hessian matrix in aerodynamic inverse design using continuous adjoint formulations*, Computers and Fluids, 37 (2008), pp. 1029–1039.

[30] F.-E. PLESSIX, *A review of the adjoint-state method for computing the gradient of a functional with geophysical applications*, Geophys. J. Int., 167 (2006), pp. 495–503.

[31] M. P. RUMPFKEIL AND D. J. MAVRIPLIS, *Efficient Hessian Calculations using Automatic Differentiation and the Adjoint Method with applications*, AIAA Journal, 48 (2010), pp. 2406–2417.

[32] S. RUUTH, *Implicit-explicit methods for reaction-diffusion problems in pattern-formation*, Journal of Mathematical Biology, 34 (1995), pp. 148–176.

[33] J. M. SANZ-SERNA, *Sympletic Runge-Kutta Schemes for Adjoint Equations, Automatic Differentiation, Optimal Control and more*, SIAM Review, 58 (2015), pp. 3–33.

[34] J. M. SANZ-SERNA, J. G. VERWER, AND W. H. HUNDSDORFER, *Convergence and order reduction of Runge-Kutta schemes applied to evolutionary problems in partial differential equations*, Numerische Mathematik, 50 (1986), pp. 405–418.

[35] C. V. STEWART, *Robust parameter estimation in computer vision*, SIAM Review, 41 (1999), pp. 513–537.

[36] A. TARANTOLA, *Inverse Problem Theory and Methods for Model Estimation*, SIAM, 2004.

[37] A. N. TIKHONOV AND V. Y. ARSENIN, *Solution of Ill-posed Problems*, Winston & Sons, 1977.

[38] A. WALTHER, *Automatic differentiation of explicit Runge-Kutta methods for optimal control*, Computational Optimization and Applications, 36 (2007), pp. 83–108.