

Derivatives Across Adaptive-Step Integration Accurately and Efficiently

Kaushik Ponnappalli* and Robert D. Falck[†]
NASA Glenn Research Center, Cleveland, OH, 44233

The numerical integration of ordinary differential equations (ODEs) is common in trajectory optimization. Previous studies have shown that the use of adaptive-step Runge-Kutta integration schemes can introduce errors in the derivatives due to non-differentiable behavior within the integration scheme itself. The authors previously demonstrated the use of an implicit adaptation to the Runge-Kutta methods to mitigate this issue, but at a significant computational cost. In this work, the authors demonstrate an explicit adaptation to the Runge-Kutta methods to address the issue that does not incur a significant computational cost.

I. Introduction

NUMERICAL integration of ODEs is a subject of interest in a number of fields, particularly in trajectory optimization, whether as a stand-alone problem or embedded in a multidisciplinary optimization. Obtaining derivatives of the output of such an integration is also of interest when application of gradient-based optimization is desired [1].

A variety of approaches to computing the derivatives have been put forth in the literature [2–6]. References [2, 3] use a *direct* or a *forward* approach while references [4–6] use an *adjoint* or a *reverse* approach. In the direct approach, the ODE is integrated and the derivatives are computed simultaneously. Whereas in the adjoint approach, the ODE is integrated and then a backward propagation of the ODE is used to compute the derivatives.

Another distinguishing factor is whether the derivatives computed are that of the ODE being integrated or those corresponding to the system that is numerically propagated. In the former approach, additional differential equations are constructed to represent the flow of the derivatives, which are then numerically integrated. This represents a "differentiate-then-discretize" approach and is used in [3–5]. The alternative approach to this, as presented in [2, 6], is to treat the numerical integration of an ODE as a series of discrete processes and to compute the derivatives of the numerical computation being performed, a "discretize-then-differentiate" approach. In [6], the author notes that this approach is better for applications involving numerical optimizers or solvers since the derivatives are those of the actual function being minimized or solved for. Whereas the differentiate-then-discretize approaches are better suited to sensitivity analyses.

In a discretize-then-differentiate approach, the integration method used naturally has an impact on how the derivatives are computed. In [3] it was shown that using a fixed-step integration method yields more accurate derivatives than using an adaptive-step method. In [2], these authors identified the source of the error in adaptive-step computed derivatives as the potential alternation between implicit and explicit behavior when computing the size of the time step. This error was significantly reduced by enforcing purely implicit behavior when computing the time step size. However, doing so caused a large increase in the computation time, thereby eliminating much of the benefit of using an adaptive-step method.

In this paper, we will explore a different modification of adaptive-step to improve the accuracy of derivative computation when using a direct discretize-then-differentiate approach. Additionally, we will summarize direct and adjoint versions of the differentiate-then-discretize methods and compare these against the new approach proposed.

II. Theory

A. Numerical Integration of an ODE

Suppose we would like to integrate some ODE that provides the rates of change of the state variables $\dot{x}$ as a function of the current state x , the current time t , and some set of parameters θ .

*Aerospace Engineer, HX5 LLC, kaushik.s.ponnappalli@nasa.gov, AIAA Member

[†]Aerospace Engineer, Mission Analysis and Architecture Branch, robert.d.falck@nasa.gov, AIAA Senior Member.

$$\dot{x} = f(x, t, \theta) \quad (1)$$

where f is the dynamics function. In the course of our analysis, we need to integrate this equation starting from some initial time t_0 for some duration of time t_d ,

$$x_f = x_0 + \int_{t_0}^{t_0+t_d} f(x, t, \theta) dt \equiv \mathcal{I}(z) \quad (2)$$

where x_f is the final state and x_0 is the initial state. Here we treat the integration of the ODE as a function $\mathcal{I}$ and all the parameters of the integration function are packaged into some single array-valued argument z .

$$z = \begin{bmatrix} x & t_0 & t_d & \theta \end{bmatrix}^T \quad (3)$$

In many dynamical systems of interest an analytic solution is not known and x_f is obtained by numerically integrating Eq. (1). There are a variety of options available for numerical integration schemes but in this paper we focus on Runge-Kutta (RK) methods. RK methods are well known but the general form of such methods is detailed here for completeness. For a fixed-step RK method, the equations of the step update are

$$x_q = x_{q-1} + h_{q-1} \sum_{i=0}^s b_i k_{qi} \quad (4)$$

$$t_q = t_{q-1} + h_{q-1} \quad (5)$$

Where s , the number of stages, is a property of the specific RK method applied and k , the state rate at each stage of the given RK method, is computed as

$$T_i = t_{q-1} + c_i h_{q-1} \quad (6)$$

$$X_i = x_{q-1} + h_{q-1} \sum_{j=0}^{i-1} a_{ij} k_{qj} \quad (7)$$

$$k_{qi} = f(T_i, X_i, \theta) \quad (8)$$

with T_i and X_i being the time and state respectively at the intermediate stages. For a fixed-step integration scheme, the size of the time step, h , is computed as

$$h = \frac{t_d}{N} \quad (9)$$

where N is the pre-specified number of steps. The a , b , and c terms are coefficients of the given the RK scheme. The coefficients for any given RK scheme are often laid out in a *Butcher tableau*.

$$\begin{array}{c|cccc} c_0 & a_{11} & a_{12} & \dots & a_{1s} \\ c_1 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array} \quad (10)$$

Adaptive-step integration schemes seek to minimize the total number of time steps taken while ensuring the accumulated error does not exceed some pre-specified tolerance, ϵ_{tol} [7]. If the integration error for a given time step is above the acceptable threshold, the step size is deemed too large and is rejected. Conversely, if the error is smaller than

the tolerance, and the step size is increased for the next time step. This is done by computing the step using two methods, one of order p and one of order $p + 1$, and the error is then computed as the difference between the two steps. In order to achieve this, the Butcher Tableau is augmented with an additional coefficient, b^* , corresponding to the higher-order step

$$\begin{array}{c|ccccc}
 c_0 & a_{11} & a_{12} & \dots & a_{1s} \\
 c_1 & a_{21} & a_{22} & \dots & a_{2s} \\
 \vdots & \vdots & \vdots & \ddots & \vdots \\
 c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\
 \hline
 & b_1 & b_2 & \dots & b_s \\
 & b_1^* & b_2^* & \dots & b_s^*
 \end{array} \tag{11}$$

In addition, to the state computed in Eq. (4), an additional state, x^* is computed as

$$x_q^* = x_{q-1} + h_{q-1} \sum_{i=0}^s b_i^* k_{qi} \tag{12}$$

and the error at the step, Δ is computed as the magnitude of the difference of Eq. (4) and Eq. (12)

$$\Delta = \left\| \sum_{i=0}^s (b_i^* - b_i) k_{qi} \right\| \tag{13}$$

In this paper, we use the equivalent expression

$$\Delta = \sqrt{\left((b^* - b)^T k_q \right)^2} \tag{14}$$

At each step, the computed error must be below some specified tolerance, ϵ_{tol} . If the error is larger than the tolerance, then the step is rejected. Regardless of whether the step is rejected, the step size is adjusted based on the error. The step is updated heuristically based on the error and the tolerance. The update used here is

$$h_q = 0.84 h_{q-1} \left(\frac{\epsilon_{tol}}{\Delta} \right)^{1/4} \tag{15}$$

If the computed step size causes the updated time to overshoot the specified final time, the step size is simply re-computed as

$$h_q = t_0 + t_d - t_{q-1} \tag{16}$$

The integration is terminated when $t_q = t_0 + t_d$.

B. Computing Derivatives Across Numerical Integration

We seek the derivatives of the final state vector, x_f and, in the case of time-varying systems, the final time, t_f , with respect to the inputs, z . One approach to computing these derivatives is to propagate them alongside the propagation of state, equivalent to the forward mode in algorithmic differentiation. This is the direct method. The other approach, akin to reverse mode in algorithmic differentiation, is to propagate the ODE forward in time to compute the states and then backward in time to compute the derivatives. This approach is also known as the adjoint method. Within each of these approaches there are two further subsets of methods to compute derivatives across ODEs; differentiate-then-discretize and discretize-then-differentiate.

In the direct method, using a differentiate-then-discretize approach yields the so-called variational equations. These equations are then propagated in tandem with the states to obtain the derivatives with respect to the inputs [3, 8, 9]. In astrodynamics applications, the derivative matrix of the final state with respect to the initial state is typically referred to as the state transition matrix [10]. The discretize-then-differentiate approach was examined in [2]. Reference [3] includes a brief treatment of the differences in the two approaches but does not fully capture the nuance associated with

computing derivatives when a time step is computed and then rejected. This is the subject of further examination in II.B.2.

In the adjoint method, adjoint equations are derived for the dynamical system of interest. In a differentiate-then-discretize approach, these equations are propagated backwards in time to obtain the desired derivatives [4]. In the discretize-then-differentiate case, the adjoint equations are applied at each time step and the chain rule is used to obtain the derivative of the final state with respect to the initial state [6].

In the following sections, we will outline the both of the direct methods of computing derivatives. In the case of the direct discretize-then-differentiate method, we will describe a modification to the traditional RK integration scheme that will greatly improve derivative accuracy. Additionally, we will also explore the differentiate-then-discretize version of the adjoint method.

1. Direct Differentiate-then-Discretize

For an ODE given by Eq. (1) with inputs, z , we can construct an augmented dynamical system [11]

$$\begin{bmatrix} \dot{x} \\ \dot{t}_0 \\ \dot{t}_d \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} f(x, t, \theta) \\ 0 \\ 0 \\ 0 \end{bmatrix} = F(x, t, \theta) \quad (17)$$

with the initial conditions

$$\begin{bmatrix} x_0 & t_0 & t_d & \theta \end{bmatrix}^T \quad (18)$$

This can be re-written as

$$\dot{Z}(t) = F(Z, t), \quad Z(t_0) = z \quad (19)$$

Following the procedure described in Eqs. (11)-(17) of [11], we can obtain equations of motion for the derivatives of the state with respects to the inputs as

$$\frac{d}{dt} \left[\frac{dx}{dz} \right] = \left[\frac{\partial f}{\partial Z} \right] \left[\frac{dZ}{dz} \right] \quad (20)$$

These equations may now be integrated numerically, In practice, this may be done in one of two ways. Firstly, they are treated as a part of the state vector and used in the error minimization process described in II.A. This is the so-called augmented state approach with F being the augmented dynamics function. The alternative is to exclude these equations from the iteration on the step size and only use the state vector when computing the error. In this paper we will only consider the augmented state approach.

2. Direct Discretize-then-Differentiate

In the discretize-then-differentiate approach, we take the derivative of the discretized update equations Eq. (4), Eq. (5), and Eq. (15) to obtain update equations for the derivatives of the final state with respect to the inputs.

$$\left[\frac{dx_q}{dz} \right] = \left[\frac{\partial x_q}{\partial z} \right] + \left[\frac{\partial x_q}{\partial x_{q-1}} \right] \left[\frac{dx_{q-1}}{dz} \right] + \left[\frac{\partial x_q}{\partial h_{q-1}} \right] \left[\frac{dh_{q-1}}{dz} \right] + \sum_{i=0}^s \left[\frac{\partial x_q}{\partial k_{qi}} \right] \left[\frac{dk_{qi}}{dz} \right] \quad (21)$$

$$\left[\frac{dt_q}{dz} \right] = \left[\frac{\partial t_q}{\partial z} \right] + \left[\frac{\partial t_q}{\partial t_{q-1}} \right] \left[\frac{dt_{q-1}}{dz} \right] + \left[\frac{\partial t_q}{\partial h_{q-1}} \right] \left[\frac{dh_{q-1}}{dz} \right] \quad (22)$$

$$\left[\frac{dh_q}{dz} \right] = \left[\frac{\partial h_q}{\partial z} \right] + \left[\frac{\partial h_q}{\partial h_{q-1}} \right] \left[\frac{dh_{q-1}}{dz} \right] + \left[\frac{\partial h_q}{\partial k_{qi}} \right] \left[\frac{dk_{qi}}{dz} \right] \quad (23)$$

The detailed derivation for each of the values may be found in [2]. However, these update equations are insufficient. As noted in [2], there is an error in these update equations stemming from steps that are rejected. That is, when the error

is larger than allowed, the size of the time step is reduced, and the update is re-computed. When this rejection and re-computation is performed at a given time step, the integrator is in essence attempting to converge the residual of a series of implicit equations, exhibiting implicit behavior locally. When the step is not rejected, the integration process is purely explicit at that given time step. This alternation of implicit and explicit behavior was found to be the cause of inaccuracies in the derivatives.

This framing is helpful in analyzing the problem when we think of adaptive-step methods as finding an "optimal" step size and motivated the solution proposed in [2]. An alternate framing of the problem is to view the numerical integration equations Eq. (4), Eq. (5) and Eq. (15) as a discrete dynamical system. When viewed through this lens, we can see there are time steps that are not captured when a step is rejected for being too large. Specifically, time steps where the step size is zero are missing. A more accurate representation of Eq. (4), Eq. (5) and Eq. (15) is

$$x_q = x_{q-1} + H(\epsilon - \Delta) h_{q-1} \sum_{i=0}^s b_i k_{qi} \quad (24)$$

$$t_q = t_{q-1} + H(\epsilon - \Delta) h_{q-1} \quad (25)$$

$$h_q = 0.84 h_{q-1} \left(\frac{\epsilon}{\Delta} \right)^{1/4} \quad (26)$$

where $H(\cdot)$ is the heaviside step function

$$H(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \quad (27)$$

This modification captures the behavior of the "zero" time steps. However, $H(x)$ is not continuous. So, we instead use the differentiable approximation

$$\eta = \frac{1}{2} + \frac{1}{2} \tanh \left(\frac{-K(\Delta - \epsilon_{tol})}{2} \right) \quad (28)$$

where K is coefficient that determines the smoothness of the approximating curve. For this application where we are attempting to mimic the behavior of step function, we may select arbitrarily large values for K . Replacing $H(x)$ with Eq. (28) in Eq. (24)-Eq. (25), we can obtain

$$x_q = x_{q-1} + \eta h_{q-1} \sum_{i=0}^s b_i k_{qi} \quad (29)$$

$$t_q = t_{q-1} + \eta h_{q-1} \quad (30)$$

with the step size update unchanged. Differentiating these equations yields

$$\left[\frac{dx_q}{dz} \right] = \left[\frac{\partial x_q}{\partial z} \right] + \left[\frac{\partial x_q}{\partial x_{q-1}} \right] \left[\frac{dx_{q-1}}{dz} \right] + \left[\frac{\partial x_q}{\partial h_{q-1}} \right] \left[\frac{dh_{q-1}}{dz} \right] + \left[\frac{\partial x_q}{\partial \eta} \right] \left[\frac{d\eta}{dz} \right] + \sum_{i=0}^s \left[\frac{\partial x_q}{\partial k_{qi}} \right] \left[\frac{dk_{qi}}{dz} \right] \quad (31)$$

$$\left[\frac{dt_q}{dz} \right] = \left[\frac{\partial t_q}{\partial z} \right] + \left[\frac{\partial t_q}{\partial t_{q-1}} \right] \left[\frac{dt_{q-1}}{dz} \right] + \left[\frac{\partial t_q}{\partial \eta} \right] \left[\frac{d\eta}{dz} \right] + \left[\frac{\partial t_q}{\partial h_{q-1}} \right] \left[\frac{dh_{q-1}}{dz} \right] \quad (32)$$

These equations may be propagated alongside the state vector. Unlike in the differentiate-then-discretize approach, these update equations never need to be included in the error minimization process.

The derivatives computed in this, the discretize-then-differentiate approach, take into consideration derivatives of the numerical integration process itself. As such, the derivatives computed in this approach will differ from those computed when following the differentiate-then-discretize. As pointed out in [6], the derivatives of the discretized process are a more accurate representation of the derivatives of the numerically integrated state since they capture the exact operations performed to obtain that final state. The derivatives obtained from propagating differentiated equations will more accurately capture the sensitivities of the analytical ODE solution.

3. Adjoint Method

The adjoint method to computing derivatives detailed here is the differentiate-then-discretize approach, also known as a *continuous adjoint* or the *adjoint sensitivity method* [12]. We take the ODE given by Eq. (1) and then define the adjoint states

$$\alpha(t) = \frac{d J_1}{d Z(t)} \quad (33)$$

$$\beta(t) = \frac{d J_2}{d Z(t)} \quad (34)$$

Here α and β are the adjoints corresponding to the state and time and $Z(t)$ is the augmented state defined in Eq. (19). The functions J_1 and J_2 typically represent cost functions that are to be minimized but in this case are the outputs we seek the derivatives of. So

$$J_1 = x(t_f) \quad (35)$$

$$J_2 = t_f \quad (36)$$

These adjoints in turn have the equations of motion

$$\frac{d \alpha}{d t} = -\alpha(t) \frac{\partial F(Z, t)}{\partial Z(t)} \quad (37)$$

$$\frac{d \beta}{d t} = -\beta(t) \frac{\partial F(Z, t)}{\partial Z(t)} \quad (38)$$

where $F(Z, t)$ is the rate of change of the augmented state. From Eq. (33) and Eq. (35), we can see it is trivial to compute the values of α at t_f once the original ODE is integrated. We can also note that this value at t_0 corresponds to the desired derivatives with respect to the state. With this information we can determine the outputs and their derivatives with respect to the inputs.

The first step is to integrate the ODE given by Eq. (1) forward in time to determine x_f and t_f . Next, we use these outputs to compute the values of $\alpha(t_f)$ and $\beta(t_f)$. We then integrate the adjoint ODEs Eq. (37) and Eq. (38) backwards in time, that is, starting at t_f and ending at t_0 to determine $\alpha(t_0)$ and $\beta(t_0)$. From Eq. (19), Eq. (33), and Eq. (35) we can show that

$$\alpha(t_0) = \frac{d x_f}{d z} \quad (39)$$

Similarly,

$$\beta(t_0) = \frac{d t_f}{d z} \quad (40)$$

The integration forward and backward in time may be performed using either the equation set of Eq. (4), Eq. (5), and Eq. (15) or Eq. (24), Eq. (25), and Eq. (26). Both methods yield the same results. In the test cases presented the modified algorithm presented in II.B.2 is used for the integration.

III. Applications

In order to demonstrate the advantages of the discretize-then-differentiate approach presented in II.B.2, we will examine two test cases. The first test problem will study the propagation of a spacecraft under the two-body model. This example, while simple, presents a motivating case for the use of adaptive-step integration. The second is the well-known van der Pol oscillator. As we will see, the dynamics of the van der Pol oscillator present a challenge for certain methods and represents a stressing case. In both problems, the derivatives will be propagated in the direct differentiate-then-discretize approach as described in II.B.1, the direct discretize-then-differentiate approach proposed in II.B.2, and the adjoint approach described in II.B.3. The accuracy of the derivatives will be evaluated by comparing against derivatives obtained via complex-step differentiation (CSD) [13] and by using the line search application as shown in [3]. To briefly summarize the line search application, a trajectory with some initial state x_0 is propagated to

obtain some final state $x_{f,0}$ and the derivatives of $x_{f,0}$ with respect to the inputs. Then, a perturbation δz is applied to the inputs and the trajectory is re-propagated to obtain the final state $x_{f,\text{actual}}$. This same final state is approximated using the derivatives from the initial propagation as

$$x_{f,\text{expected}} = x_{f,0} + \left[\frac{dx_{f,0}}{dz} \right] \delta z \quad (41)$$

The error is computed as the magnitude of the relative error in the final states:

$$\text{error} = \frac{|x_{f,\text{expected}} - x_{f,\text{actual}}|}{|x_{f,\text{actual}}|} \quad (42)$$

All propagations are performed using a Runge-Kutta-Fehlberg integrator with order 7 and embedded order 8(RK78). The integration tolerance, ϵ_{tol} , used is 10^{-9} . For the direct discretize-then-differentiate integration, the steepness factor, K , for the heaviside approximation in Eq. (28) is 10^{15} .

A. Orbit Propagation

The test case examined here is the propagation of a two-dimensional orbit under the two-body gravity model. The equations of motion are

$$\begin{bmatrix} \dot{r}_x & \dot{r}_y & \dot{v}_x & \dot{v}_y \end{bmatrix} = \begin{bmatrix} v_x & v_y & -\frac{\mu}{(r_x^2 + r_y^2)^{3/2}} r_x & -\frac{\mu}{(r_x^2 + r_y^2)^{3/2}} r_y \end{bmatrix} \quad (43)$$

The initial conditions for the integration in normalized Length Units (LU) and Time Units (TU) are:

$$\begin{bmatrix} r_{x0} & r_{y0} & v_{x0} & v_{y0} \end{bmatrix} = \begin{bmatrix} 1 - e & 0 & 0 & \sqrt{\frac{1+e}{1-e}} \end{bmatrix} \quad (44)$$

where e is the eccentricity of the orbit. For this test case, $e = 0.8$ and the period of the orbit is 2π . The standard gravitational parameter is $\mu = 1 \text{ LU}^3/\text{TU}^2$. The orbit is propagated for a duration $t_d = 2n\pi$ where n is the number of revolutions. Different values of n are used and the impact of extended propagation on the derivatives is examined. For brevity, only the derivatives with respect to the initial state are shown here. The derivatives with respect to t_0 are identically zero since this is a time invariant system and, as shown in [2], the derivatives of t_d and μ largely follow the same trends as the derivatives with respect to the the initial state.

As a result of the integration, the derivatives with respect to the states are computed. In Fig. 1, we can see the accuracy of the predicted state using the line search application and in Fig. 2, we see the accuracy of the propagated derivatives when compared to those computed using CSD. In both instances, we find that the differentiate-then-discretize approach is less accurate than the discretize-then-differentiate approach. This appears to indicate that the modifications to the adaptive-step algorithm were successful in mitigating the issues in the traditional method.

The results obtained from the adjoint method showcase an interesting phenomenon. The derivatives are accurate to a similar order as the direct discretize-then-differentiate method when using the line search error as the evaluation metric but the error is substantially larger when using CSD as the criterion. This appears to indicate that there is path-dependent behavior in the integration process that is captured by CSD but not by the analytical adjoint equations. Ultimately, from the perspective of a numerical solver or optimizer, the line-search error is the most important metric and from that perspective the adjoint method computes accurate derivatives. It is more likely the CSD is providing erroneous derivatives but further study is warranted.

In addition to accuracy, run time is an important consideration. A previous study proposed a solution to the inaccuracy of derivative computation but did so at the cost of a significantly increased run time [2]. This is obviously undesirable as it eliminates a major advantage of using adaptive-step integration, improved run time due to the smaller amount of time steps taken. To examine the performance of different methods of propagation, we compare them to a simple state-only integration. Fig. 3 shows the computation time for each of the three methods normalized by the time taken to integrate the ODE without propagating the derivatives. This allows us to examine the additional computational cost associated with the each of the approaches. We can see that the proposed discretize-then-differentiate method outperforms the other two methods. This is as we would expect since this new method does not include the derivatives in the error computation process and can therefore take fewer time steps and achieve the same integration accuracy. We may also note that the adjoint method is significantly slower than both of the direct methods. The adjoint method suffers from two setbacks. First, the propagation must be performed twice, once forward and once backward. Second, the reverse

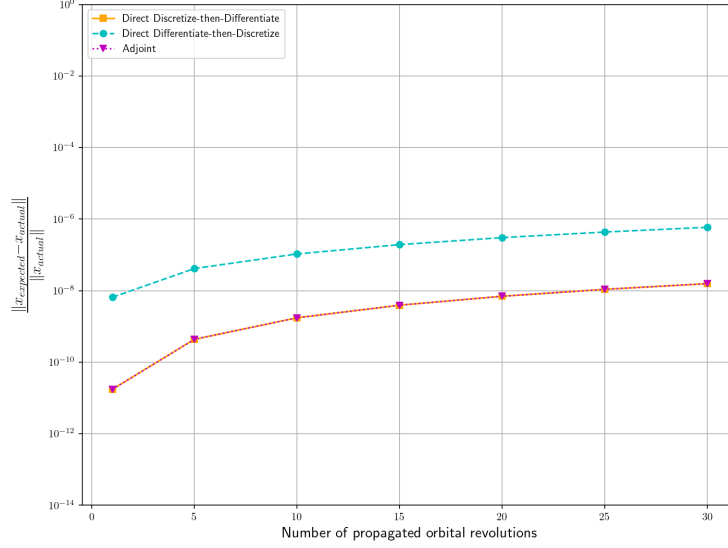


Fig. 1 Error in the final state computed using the line search application - orbit propagation

propagation consists of both the state and the adjoints meaning that more time steps must be taken to remain under the error tolerance. So while the adjoint method achieves the same accuracy as the direct discretize-then-differentiate method, it does so at a significant cost in compute time.

B. Van der Pol Oscillator

The van der Pol oscillator is a dynamical system that obeys the ODE

$$\ddot{x} + x = \mu (1 - x^2) \dot{x} \quad (45)$$

where μ is a constant parameter. This equation is put into state space form to obtain the system of equations

$$\dot{x} = y \quad (46)$$

$$\dot{y} = -x + \mu (1 - x^2) y \quad (47)$$

The van der Pol oscillator will eventually settle into a periodic motion regardless of initial conditions [14]. This can be visualized by integrating the ODEs forward in time with two differing initial conditions

$$\begin{bmatrix} x_0 & y_0 \end{bmatrix} = \begin{bmatrix} 0.5 & 1.0 \end{bmatrix}, \begin{bmatrix} -1.0 & 4.0 \end{bmatrix} \quad (48)$$

This propagation is performed for a duration 15 s. The resulting phase space diagram Fig. 4 shows the convergence of both initial conditions to the periodic solution.

To illustrate the challenge of computing derivatives for this ODE, we will use the first of the two initial conditions listed and integrate for different durations. For each duration, we will apply the line search technique for a perturbation δz of 10^{-9} . The results can be seen in Fig. 5.

We initially observe a similar trend in the results as in the orbit propagation problem, the direct differentiate-then-discretize method yields the least accurate derivatives while the adjoint and the direct discretize-then-differentiate method are at a similar accuracy. However, at a certain point the inaccuracy of the adjoint derivatives spikes and we no longer obtain useful results. This is the reason those data points are missing from Fig. 5, they are so inaccurate as to not be useful. This is due to the nature of the dynamics in the van der Pol oscillator. The dynamics converge to a limit cycle and remain in that periodic motion but when propagating backward it is not always possible to recover the exact path that forward integration follows. Even extremely small errors cause the forward and backward trajectories to diverge. Fig. 6 shows one such case. Similar issues have previously been noted in other works [15] and have been addressed using check-pointing. Consideration of such mitigation strategies is beyond the scope of this work.

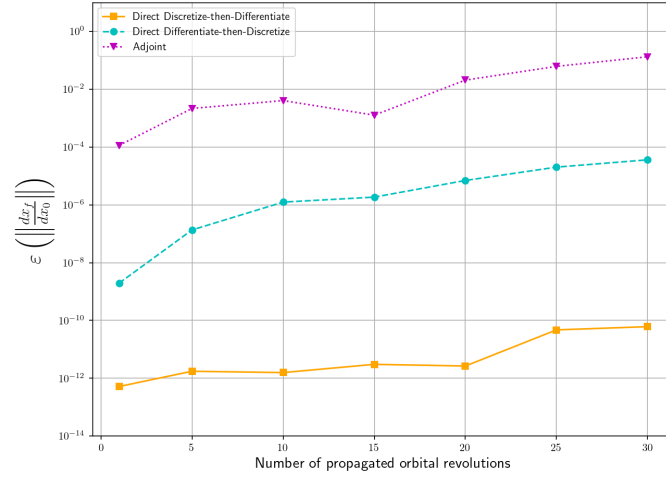


Fig. 2 Error in the derivatives compared to complex-step differentiation - orbit propagation

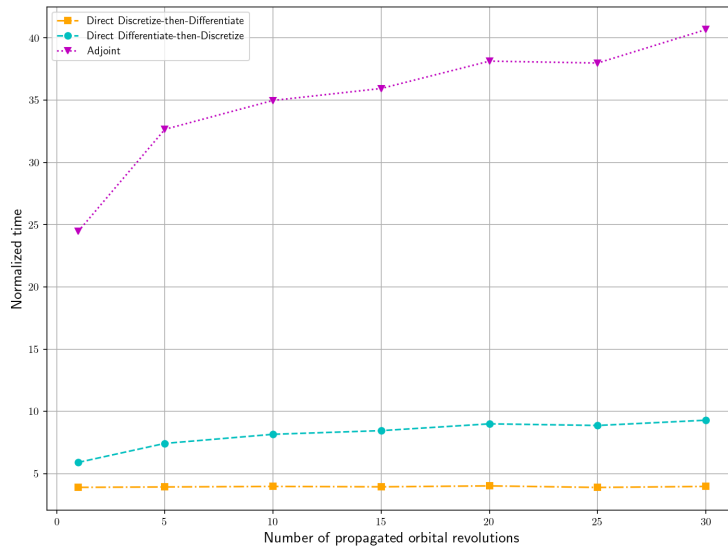


Fig. 3 Normalized compute times - orbit propagation

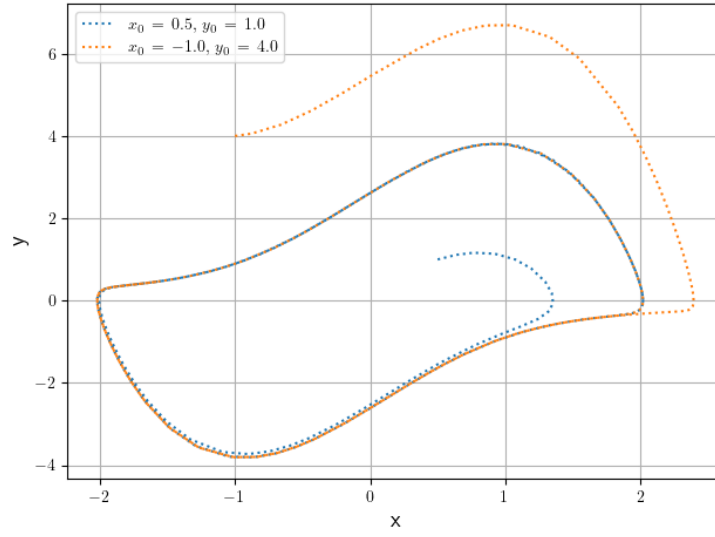


Fig. 4 Van der Pol oscillator trajectories in phase space given different initial conditions

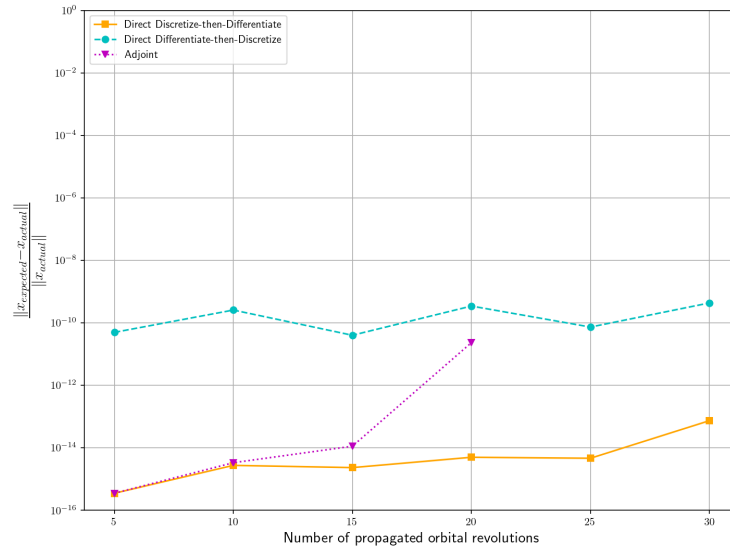


Fig. 5 Error in the final state computing using the line search application - van der Pol oscillator

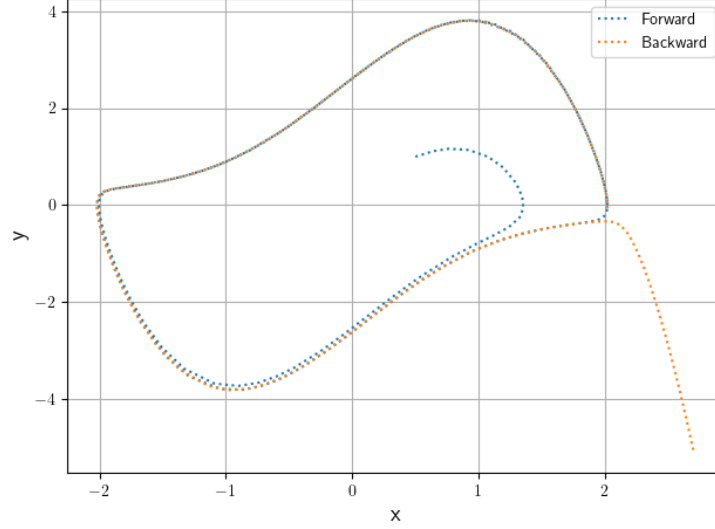


Fig. 6 Van der Pol oscillator trajectory divergence when integrating forward and backward

IV. Conclusion

In this work we have demonstrated a new method for integrating ODEs and accurately computing the derivatives of their outputs. We built upon previous work that mitigated this issue and developed approach this is both fast and accurate. This new approach eliminates the issue of inaccurate derivatives when using traditional adaptive-step methods to integrate ODEs. It eliminates the alternating implicit and explicit behavior found in the traditional approach by forcing purely explicit behavior. By forcing explicit behavior, the new approach is able to compute highly accurate derivatives without an accompanying increase in compute times and it is able to do so for sensitive and chaotic systems. Applying this approach should significantly improve the quality of derivatives obtained when integrating ODEs, enabling for better and faster trajectory optimization when using gradient-based solvers and optimizers.

Acknowledgments

This work was funded under NASA’s Transformational Tools and Technologies and Advanced Air Transport Technology projects. The authors are grateful for the support.

References

- [1] Betts, J. T., *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, Society for Industrial and Applied Mathematics, 2010.
- [2] Ponnappalli, K., and Falck, R. D., “Obtaining Accurate Derivatives Across Adaptive-Step Integration,” *AIAA SCITECH 2024 Forum*, 2024. <https://doi.org/10.2514/6.2024-0160>, URL <https://arc.aiaa.org/doi/abs/10.2514/6.2024-0160>.
- [3] Pellegrini, E., and Russell, R. P., “On the Computation and Accuracy of Trajectory State Transition Matrices,” *Journal of Guidance, Control, and Dynamics*, Vol. 39, No. 11, 2016, pp. 2485–2499. <https://doi.org/10.2514/1.G001920>, URL <https://doi.org/10.2514/1.G001920>.
- [4] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D., “Neural Ordinary Differential Equations,” 2018. <https://doi.org/https://doi.org/10.48550/arXiv.1806.07366>, URL [arXiv:1806.07366](https://arxiv.org/abs/1806.07366).
- [5] Margolis, B. W. L., “A Sweeping Gradient Method for Ordinary Differential Equations with Events,” *Journal of Optimization Theory and Applications*, 2023.
- [6] Sandu, A., “On the properties of runge-kutta discrete adjoints,” *Proceedings of the 6th International Conference on Computational Science - Volume Part IV*, Springer-Verlag, Berlin, Heidelberg, 2006, p. 550–557. https://doi.org/10.1007/11758549_76, URL https://doi.org/10.1007/11758549_76.

- [7] Fehlberg, E., *Classical Fifth-, Sixth-, Seventh-, and Eighth-order Runge-Kutta Formulas with Stepsize Control*, NASA technical report, National Aeronautics and Space Administration, 1968. URL <https://books.google.com/books?id=5Uy6HLvsZRgC>.
- [8] Eberhard, P., and Bischof, C., “Automatic Differentiation of Numerical Integration Algorithms,” *Mathematics of Computation*, Vol. 68, No. 226, 1999, pp. 717–731. <https://doi.org/https://doi.org/10.1090/S0025-5718-99-01027-3>.
- [9] Park, R. S., and Scheeres, D. J., “Nonlinear Mapping of Gaussian Statistics: Theory and Applications to Spacecraft Trajectory Design,” *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 6, 2006, pp. 1367–1375. <https://doi.org/10.2514/1.20177>, URL <https://doi.org/10.2514/1.20177>.
- [10] Battin, R. H., *An Introduction to the Mathematics and Methods of Astrodynamics*, American Institute of Aeronautics and Astronautics, 1987.
- [11] Turner, J., Majji, M., and Junkins, J., “High-Order State and Parameter Transition Tensor Calculations,” *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, 2008. <https://doi.org/10.2514/6.2008-6453>, URL <https://doi.org/10.2514/6.2008-6453>.
- [12] Pontryagin, L. S., Boltyanski, V. G., Gamkrelidze, R. V., and Mischenko, E. F., *The Mathematical Theory of Optimal Processes*, Wiley, New York, 1962.
- [13] Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., “The Complex-Step Derivative Approximation,” *ACM Trans. Math. Softw.*, Vol. 29, No. 3, 2003, p. 245–262. <https://doi.org/10.1145/838250.838251>, URL <https://doi.org/10.1145/838250.838251>.
- [14] Verhulst, F., *Nonlinear Differential Equations and Dynamical Systems*, 2nd ed., Springer, 1996.
- [15] Gholami, A., Keutzer, K., and Biros, G., “ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs,” 2019. <https://doi.org/https://doi.org/10.48550/arXiv.1902.10298>, URL [arXiv:1902.10298](https://doi.org/https://doi.org/10.48550/arXiv.1902.10298).