SANDIA REPORT

SAND2013-4299
Unlimited Release
Printed May 2013

IMEX-a: An Adaptive, Fifth Order Implicit-Explicit Integration Scheme

Matthew R. Brake

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

SAND2013-4299
Unlimited Release
Printed May 2013

IMEX-a: An Adaptive, Fifth Order Implicit-Explicit Integration Scheme

Matthew R. Brake
Component Science and Mechanics
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1070

Abstract

This report presents an efficient and accurate method for integrating a system of ordinary differential equations, particularly those arising from a spatial discretization of partially differential equations. The algorithm developed, termed the IMEX_a algorithm, belongs to a class of algorithms known as implicit-explicit (IMEX) methods. The explicit step is based on a fifth order Runge-Kutta explicit step known as the Dormand-Prince algorithm, which adaptively modifies the time step by calculating the error relative to a fourth order estimation. The implicit step, which follows the explicit step, is based on a backward Euler method, a special case of the generalized trapezoidal method. Reasons for choosing both of these methods, along with the algorithm development are presented. In applications that have less stringent accuracy requirements, several other methods are available through the IMEX_a toolbox, each of which simplify the fifth order Dormand-Prince explicit step: the third order Bogacki-Shampine method, the second order Midpoint method, and the first order Euler method. The performance of the algorithm is evaluated on to examples. First, a two pawl system with contact is modeled. Results predicted by the IMEX_a algorithm are compared to those predicted by six widely used integration schemes. The IMEX_a algorithm is demonstrated to be significantly faster (by up to an order of magnitude) and at least as accurate as all of the other methods considered. A second example, an acoustic standing wave, is presented in order to assess the accuracy of the IMEX_a algorithm. Finally, sample code is given in order to demonstrate the implementation of the proposed algorithm.

Acknowledgment

I would like to thank and acknowledge Chris Volk for his help in some of the intermediate algorithm development, Jordan Massad for driving the need for faster simulation capabilities, as well as Gil Benavides, who provided an ideal project to develop and refine the theory presented in this report, Irina Kalashnikova for providing the code for the acoustic standing wave example, Matthew Williams for commenting on drafts of this document, and the team of Behdad Beheshti, Kenny Chowdhary, Jason Davis, and Shanshan Wang for their preliminary work on adapting an implicit-explicit algorithm to problems detailed in what follows.

Contents

Figures

Tables

1 Introduction

The IMEX_a algorithm is an adaptive time stepping, error-controlled, fifth order implicit-explicit (IMEX) algorithm. The Matlab toolbox IMEX_a has been developed concurrently with several rigid body dynamic models. These models include inertial sensors [1, 2], elastic-plastic impact in mechanisms [3–6], energy dissipation across frictional interfaces [7, 8], fluid flow over panels [9, 10], and chatter in leaf spring-connector pin systems [11, 12].

Each of the problems that are mentioned above are generally classified as stiff nonlinear problems. A stiff system is characterized by having a differential equation that necessitates most numerical methods to take infinitesimally small step sizes in order to achieve numerical stability. These problems often exhibit behavior in which the numerical solution in a given regime can either oscillate about the true solution (leading to computational inefficiencies due to the changing sign of the velocity terms), or else grows unbounded. By Dahlquist's theorem [13], explicit methods are unstable when it comes to solving stiff systems. Implicit integration schemes, however, are often low accuracy methods. The IMEX_a algorithm thus seeks an efficient, stable solution for the temporal integration of stiff systems of equations.

The IMEX_a toolbox is an approach to discretizing and integrating a system of equations. Several existing approaches are currently built into Matlab, such as the 'ode' family of solvers. These solvers, though, are generally explicit methods that often do not converge for sufficiently stiff problems [2] or introduce large amounts of numerical damping. The IMEX_a algorithm, by contrast, is shown to be a more efficient approach to numerically solve a stiff, nonlinear system of differential equations, as demonstrated in Sections 4 and 5. The theoretical basis for the algorithm is first presented in Section 2, and details about the code implementation are provided in Section 3.

2 Theoretical Development

The IMEX_a algorithm is composed of two routines: a fifth order Runge-Kutta method for the explicit step, and a backward Euler method for the implicit step. The time increment for each step is adaptively determined using a Runge-Kutta fourth and fifth order Dormand-Prince method concurrently with the explicit portion of the integration scheme. The fifth order Runge-Kutta method is chosen due to its higher accuracy and ability to choose a larger stable time step than lower order methods; however, this comes at the trade-off of necessitating more evaluations of the system's force function. For less stiff problems, or cases where less stringent controls on accuracy are required, the algorithm can potentially be more efficient with a lower order method. For this reason, an option is included in the IMEX_a toolbox to use a second and third order Runge-Kutta algorithm known as the Bogacki-Shampine method for the explicit step and adaptive time stepping in place of the Dormand-Prince method. Two additional low order algorithms are also included in the toolbox for solution of linear systems: a first order Euler method, and a second order Midpoint method.

2.1 Mathematical Formalism of IMEX methods

A simpler case of an IMEX method (hereafter referred to as the first order IMEX method) is composed of a first order Runge-Kutta explicit step (an explicit Euler method) with the previously mentioned backward Euler implicit step. This case is considered in what follows to demonstrate the mathematical formalism of an IMEX method. Given a system

$$\mathbb{I}\frac{\mathrm{d}}{\mathrm{d}t}\left(\underset{\sim}{y}(t)\right) = \mathbb{A}\underset{\sim}{y}(t) + \underset{\sim}{f}\left(\underset{\sim}{y},t\right), \tag{1}$$

where $\mathbb{I}$ is the identity matrix, $\underset{\sim}{y}$ is the system's "unknown state" vector, t is time, $\mathbb{A}$ is the system's matrix, and $\underset{\sim}{f}$ is the vector of applied forces and all nonlinearities in the system (such as terms like y_i^3), the first order IMEX method is derived by taking a forward difference approximation for the time derivative

$$\mathbb{I}\left(\frac{\underset{\sim}{y}_{n+1} - \underset{\sim}{y}_n}{\triangle t}\right) = \mathbb{A}\underset{\sim}{y}_{n+1} + \underset{\sim}{f}_{n+1} \tag{2}$$

at integration step n with time step size $\triangle t$. Grouping like terms together

$$(\mathbb{I} - \triangle t\mathbb{A})\underset{\sim}{y}_{n+1} = \left(\underset{\sim}{y}_n + \triangle t\underset{\sim}{f}_{n+1}\right) \tag{3}$$

$$\underset{\sim}{y}_{n+1} = (\mathbb{I} - \triangle t\mathbb{A})^{-1}\left(\underset{\sim}{y}_n + \triangle t\underset{\sim}{f}_{n+1}\right). \tag{4}$$

Solution of this equation is then divided into two steps, which distinguishes it from a purely implicit method

$$\underset{\sim}{y}_n^* = \underset{\sim}{F}\left(\triangle t, \underset{\sim}{y}_n, \underset{\sim}{f}_{n+1}\right), \tag{5}$$

where for the first order case

$$\underset{\sim}{F}\left(\triangle t, \underset{\sim}{y}_n, \underset{\sim}{f}_{n+1}\right) = \underset{\sim}{F}^{(1)} = \left(\underset{\sim}{y}_n + \triangle t \underset{\sim}{f}_{n+1}\right), \tag{6}$$

and

$$\underset{\sim}{y}_{n+1} = (\mathbb{I} - \triangle t \mathbb{A})^{-1} \underset{\sim}{F}^{(1)}. \tag{7}$$

Equations 5 and 6 are a first order Runge-Kutta explicit integration step and Eq. 7 is a backward Euler implicit integration step. This conceptual division of Eq. 4 is the key aspect of an IMEX algorithm. While for the first order IMEX method, this is still mathematically equivalent to a generalized trapezoidal method, higher order approximations for the explicit step will offer improved computational performance. In the case where the system equation is of the form

$$\mathbb{M}\frac{\mathrm{d}}{\mathrm{d}t}\left(\underset{\sim}{y}\right) + \mathbb{K}\underset{\sim}{y} = \underset{\sim}{f}^*, \tag{8}$$

the framework established in Eq. 1 can be utilized via the identities

$$\mathbb{A} = -\mathbb{M}^{-1}\mathbb{K} \qquad \text{and} \qquad \underset{\sim}{f} = \mathbb{M}^{-1}\underset{\sim}{f}^*. \tag{9}$$

2.2 Higher Order Extensions

The Fifth Order Runge-Kutta Explicit Step

From (5) and (6), a higher order approximation of the explicit step can be obtained by direct substitution. In the IMEX_a algorithm, a fifth order Runge-Kutta method is used, which replaces Eq. 5 with

$$\underset{\sim}{F}\left(\triangle t, \underset{\sim}{y}_n, \underset{\sim}{f}_{n+1}\right) = \underset{\sim}{F}^{(5)} = \underset{\sim}{y}_n + \frac{35}{384}\underset{\sim}{\kappa}_1 + \frac{500}{1113}\underset{\sim}{\kappa}_3 + \frac{125}{192}\underset{\sim}{\kappa}_4 - \frac{187}{6784}\underset{\sim}{\kappa}_5 + \frac{11}{84}\underset{\sim}{\kappa}_6, \tag{10}$$

where

$$\begin{aligned}
\underset{\sim}{\kappa}_1 &= \triangle t \underset{\sim}{f}\left(\underset{\sim}{y}_n, t_n\right) \\
\underset{\sim}{\kappa}_2 &= \triangle t \underset{\sim}{f}\left(\underset{\sim}{y}_n + \tfrac{1}{5}\underset{\sim}{\kappa}_1, t_n + \tfrac{1}{5}\triangle t\right) \\
\underset{\sim}{\kappa}_3 &= \triangle t \underset{\sim}{f}\left(\underset{\sim}{y}_n + \tfrac{3}{40}\underset{\sim}{\kappa}_1 + \tfrac{9}{40}\underset{\sim}{\kappa}_2, t_n + \tfrac{3}{10}\triangle t\right) \\
\underset{\sim}{\kappa}_4 &= \triangle t \underset{\sim}{f}\left(\underset{\sim}{y}_n + \tfrac{44}{45}\underset{\sim}{\kappa}_1 - \tfrac{56}{15}\underset{\sim}{\kappa}_2 + \tfrac{32}{9}\underset{\sim}{\kappa}_3, t_n + \tfrac{4}{5}\triangle t\right) \\
\underset{\sim}{\kappa}_5 &= \triangle t \underset{\sim}{f}\left(\underset{\sim}{y}_n + \tfrac{19372}{6561}\underset{\sim}{\kappa}_1 - \tfrac{25360}{2187}\underset{\sim}{\kappa}_2 + \tfrac{64448}{6561}\underset{\sim}{\kappa}_3 - \tfrac{212}{729}\underset{\sim}{\kappa}_4, t_n + \tfrac{8}{9}\triangle t\right) \\
\underset{\sim}{\kappa}_6 &= \triangle t \underset{\sim}{f}\left(\underset{\sim}{y}_n + \tfrac{9017}{3168}\underset{\sim}{\kappa}_1 - \tfrac{355}{33}\underset{\sim}{\kappa}_2 + \tfrac{46732}{5247}\underset{\sim}{\kappa}_3 + \tfrac{49}{176}\underset{\sim}{\kappa}_4 - \tfrac{5103}{18656}\underset{\sim}{\kappa}_5, t_n + \triangle t\right) \\
\underset{\sim}{\kappa}_7 &= \triangle t \underset{\sim}{f}\left(\underset{\sim}{F}^{(5)}, t_n + \triangle t\right)
\end{aligned} \tag{11}$$

and their coefficients in (10) are found via the extended Butcher Tableau [14, 15]. Equation 10 is the Dormand-Prince [14] fifth order accurate solution to the ordinary differential equations used to model the system (1). The Dormand-Prince method is a form of the Runge-Kutta explicit integration equations, and is specifically chosen as it is a more suitable solution method for fifth order approximations than other, similar methods such as the Fehlberg or Cash-Karp methods [15]. The quantity κ_7 is used in the adaptive time step calculation as discussed in the following section.

The Dormand-Prince Adaptive Time Stepping Method

The time step for the IMEX_a numerical integration scheme is determined by first calculating the error between the fifth order estimate of $\underset{\sim}{F}^{(5)}$ (given in Eq. 10), and the fourth order estimate

$$\underset{\sim}{F}^{(4)} = \underset{\sim}{y}_n + \frac{5179}{57600}\underset{\sim}{\kappa}_1 + \frac{7571}{16695}\underset{\sim}{\kappa}_3 + \frac{393}{640}\underset{\sim}{\kappa}_4 - \frac{92097}{339200}\underset{\sim}{\kappa}_5 + \frac{187}{2100}\underset{\sim}{\kappa}_6 + \frac{1}{40}\underset{\sim}{\kappa}_7. \tag{12}$$

Thus, the error function is

$$\begin{aligned} \underset{\sim}{e}_n &= \left| \underset{\sim}{F}^{(5)} - \underset{\sim}{F}^{(4)} \right| \\ \underset{\sim}{e}_n &= \left| \left(\frac{35}{384} - \frac{5179}{57600} \right) \underset{\sim}{\kappa}_1 + \left(\frac{500}{1113} - \frac{7571}{16695} \right) \underset{\sim}{\kappa}_3 + \left(\frac{125}{192} - \frac{393}{640} \right) \underset{\sim}{\kappa}_4 \right. \\ &\quad \left. - \left(\frac{187}{6784} - \frac{92097}{339200} \right) \underset{\sim}{\kappa}_5 + \left(\frac{11}{84} - \frac{187}{2100} \right) \underset{\sim}{\kappa}_6 - \left(\frac{1}{40} \right) \underset{\sim}{\kappa}_7 \right|. \end{aligned} \tag{13}$$

The error criteria is then calculated as

$$\chi = \left(\frac{\tau}{\text{norm}\left(\underset{\sim}{e}_n \right)} \right)^{1/4}, \tag{14}$$

for a given error tolerance τ. Equations 10 and 13 are iteratively solved, varying $\triangle t$, until $\chi > (1/2)^{1/4}$ (i.e. until the norm of the error is less than twice the specified error tolerance). In each iteration, the new time step is calculated as

$$\triangle t \equiv \chi \, \triangle t. \tag{15}$$

In order to prevent numerical oscillations, $\triangle t$ is limited to only change by $\pm 10\%$, provided that the error is within the maximum specified tolerance τ.

The Bogacki-Shampine Second/Third Order Method

In cases where a lower accuracy solution is acceptable or for a less stiff nonlinear system, some computational savings can be introduced by means of using a lower order explicit method. The

Dormand-Prince method, a fifth order accurate method, requires seven evaluations of the system's force function at every time step. By contrast, the third order accurate Bogacki-Shampine method requires only four force function evaluations at every time step. Similarly to the Dormand-Prince method, the Bogacki-Shampine method chooses the time step size by comparing the error between two different explicit estimations of the solution. The third order accurate solution is given as

$$\underset{\sim}{F}\left(\triangle t, \underset{\sim}{y}_n, \underset{\sim}{f}_{n+1}\right) = \underset{\sim}{F}^{(3)} = \underset{\sim}{y}_n + \frac{2}{9}\underset{\sim}{\kappa}_1 + \frac{1}{3}\underset{\sim}{\kappa}_2 + \frac{4}{9}\underset{\sim}{\kappa}_3, \tag{16}$$

with coefficients

$$\begin{aligned}
\underset{\sim}{\kappa}_1 &= \triangle t \underset{\sim}{f}\left(\underset{\sim}{y}_n, t_n\right) \\
\underset{\sim}{\kappa}_2 &= \triangle t \underset{\sim}{f}\left(\underset{\sim}{y}_n + \tfrac{1}{2}\,\triangle t \underset{\sim}{\kappa}_1, t_n + \tfrac{1}{2}\,\triangle t\right) \\
\underset{\sim}{\kappa}_3 &= \triangle t \underset{\sim}{f}\left(\underset{\sim}{y}_n + \tfrac{3}{4}\,\triangle t \underset{\sim}{\kappa}_2, t_n + \tfrac{3}{4}\,\triangle t\right) \\
\underset{\sim}{\kappa}_4 &= \triangle t \underset{\sim}{f}\left(\underset{\sim}{F}^{(3)}{}_n, t_n + \triangle t\right),
\end{aligned} \tag{17}$$

where the weights in Eq. 16 are determined by the extended Butcher Tableau [14, 15]. The second order accurate solution, used for determining the relative error and estimating the next $\triangle t$ is

$$\underset{\sim}{F}^{(2)} = \underset{\sim}{y}_n + \frac{7}{24}\underset{\sim}{\kappa}_1 + \frac{1}{4}\underset{\sim}{\kappa}_2 + \frac{1}{3}\underset{\sim}{\kappa}_3 + \frac{1}{8}\underset{\sim}{\kappa}_4. \tag{18}$$

Similarly to the Dormand-Prince method,

$$\underset{\sim}{e}_n = \left|\underset{\sim}{F}^{(3)} - \underset{\sim}{F}^{(2)}\right|, \tag{19}$$

and the same criteria are used for calculating the time step size

$$\chi = \left(\frac{\tau}{\mathrm{norm}(\underset{\sim}{e}_n)}\right)^{1/4}, \tag{20}$$

$$\triangle t \equiv \chi\, \triangle t. \tag{21}$$

The Euler-Midpoint First/Second Order Method

In order to provide an efficient methodology for linear systems, two other methods are available in the toolbox: a first order Euler method, and a second order Midpoint method. It is advantageous to have a simple and efficient method to apply when the system contains no nonlinearities (i.e. is purely linear); however, even some linear systems still are better solved using the higher order algorithms. With those considerations in mind, these two methods are included, and adaptive time

stepping is determined as the error between these two estimates. The first order estimate is given in Eq. 6, and the second order estimate is

$$\underset{\sim}{F}^{(2)} = \underset{\sim}{y}_n + \triangle t \underset{\sim}{f}\left(\underset{\sim}{y}_n + \frac{1}{2}\triangle t \underset{\sim}{F}^{(1)}, t_n + \frac{1}{2}\triangle t\right). \tag{22}$$

Similarly to the methods discussed previously,

$$\underset{\sim}{e}_n = \left|\underset{\sim}{F}^{(2)} - \underset{\sim}{F}^{(1)}\right|, \tag{23}$$

and the same criteria are used for calculating the time step size

$$\chi = \left(\frac{\tau}{\text{norm}(\underset{\sim}{e}_n)}\right)^{1/4}, \tag{24}$$

$$\triangle t \equiv \chi \triangle t. \tag{25}$$

Alternative Explicit Methods

Other methods (see [14, 15]) were investigated as alternatives to the Dormand-Prince and the Bogacki-Shampine methods as well as the Euler and Midpoint methods, such as the second order Runge-Kutta method referred to as Heun's method, the fourth order Runge-Kutta method referred to as Fehlberg's method, the fourth order Adams-Bashforth method, and the second order Adams-Bashforth-Moulton method. The last two methods, attributed to Adams, Bashforth, and Moulton, belong to a separate family of numerical algorithms from the Runge-Kutta family; however, they are also explicit methods for integrating systems of equations. For each of the methods considered, the numerical stability was not as high as for the Dormand-Prince and Bogacki-Shampine methods, with the exception of Fehlberg's method; however, Fehlberg's method does not offer any significant advantages over either the Dormand-Prince or the Bogacki-Shampine methods, and is not discussed here. Thus, the final version of the toolbox only employs the four explicit methods detailed in the previous sections.

2.3 Implicit Integration Step Comments

In Eqs. 4 and 7, a backward Euler method implicit integration step is utilized. This method belongs to a class of schemes termed the generalized trapezoidal method [13], and is a linear, multi-step method. Dahlquist's theorem [13], makes three statements regarding linear multi-step methods

1. An explicit, unconditionally stable linear multi-step method does not exist.

2. A third order accurate, unconditionally stable linear multi-step method does not exist.

3. The second order accurate, unconditionally stable linear multi-step method with the smallest local truncation error is the trapezoidal rule.

Stability for the trapezoidal method requires that for a given period T, with numerical period error $\bar{T} - T$,

$$\frac{\bar{T} - T}{T} \leq 0.05, \tag{26}$$

which is equivalent to

$$\frac{\triangle t}{T} \leq 0.125.$$

This inequality must hold true for the highest frequency of interest. Thus, for a given frequency ω_n, the period is $T_n = 2\pi/\omega_n$, and the time step must satisfy

$$\frac{\triangle t}{T_n} \leq 0.125. \tag{27}$$

The general trapezoidal method, in its most general form, seeks solutions to

$$\mathbb{M}\frac{\mathrm{d}}{\mathrm{d}t}\left(\underset{\sim}{y}\right) + \mathbb{K}\underset{\sim}{y} = \underset{\sim}{f}\left(\underset{\sim}{y}, t\right). \tag{28}$$

Note that this is the same as Eq. 8. For systems where the independent variable is time, the terms $\frac{\mathrm{d}}{\mathrm{d}t}\left(\underset{\sim}{y}\right)$ and $\underset{\sim}{y}$ respectively correspond to the system's velocity and displacement. There are two approaches for the solution of Eq. 28: the velocity form and the displacement form. The goal of each form is to eliminate either the displacement or the velocity from the system of equations by estimating it in terms of the velocity or displacement, respectively. The displacement form of the solution, given in what follows, is a more efficient method when $\mathbb{M}$ is diagonal, which is the case for the IMEX_a framework. For systems with a non-diagonal matrix $\mathbb{M}$, the system can be forced to be block diagonal through the use of the common approach of mass lumping or by multiplying (28) by $\mathbb{M}^{-1}$. Assuming that the displacement and velocity of the system is known at time step n, the state of the system is estimated for the next time step

$$\underset{\sim}{y}^*_{n+1} = \underset{\sim}{y}_n + (1-\alpha)\,\triangle t \frac{\mathrm{d}}{\mathrm{d}t}\left(\underset{\sim}{y}_n\right). \tag{29}$$

Second, the displacement at the next time step $(n+1)$ is calculated via

$$\underset{\sim}{y}_{n+1} = \alpha\,\triangle t\left(\mathbb{M} + \alpha\,\triangle t\mathbb{K}\right)^{-1}\left(\underset{\sim}{f}_{n+1} + \frac{1}{\alpha\,\triangle t}\mathbb{M}\underset{\sim}{y}^*_{n+1}\right). \tag{30}$$

The velocity at time step $n+1$ is then found via a backward difference

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\underset{\sim}{y}_{n+1}\right) = \frac{\underset{\sim}{y}_{n+1} - \underset{\sim}{y}^*_{n+1}}{\alpha\,\triangle t}. \tag{31}$$

This form further simplifies when the velocity vector is not needed, which can introduce further computational savings. In this case, the displacement vector can be directly calculated as

$$\underset{\sim}{y}_{n+1} = (\mathbb{M} + \alpha \triangle t\mathbb{K})^{-1} \left((\mathbb{M} - (1-\alpha) \triangle t\mathbb{K}) \underset{\sim}{y}_n + \triangle t \left(\alpha \underset{\sim}{f}_{n+1} + (1-\alpha) \underset{\sim}{f}_n \right) \right). \tag{32}$$

The selection of α is discussed in [13]. For the case where $(\lambda \triangle t) \to 1$, where λ is the largest eigenvalue of the system, $\alpha \to 1$, and Eq. 32 simplifies to

$$\underset{\sim}{y}_{n+1} = (\mathbb{M} + \triangle t\mathbb{K})^{-1} \left(\mathbb{M} \underset{\sim}{y}_n + \triangle t \underset{\sim}{f}_{n+1} \right). \tag{33}$$

Recasting this equation in the form of (1), with $\mathbb{M} = \mathbb{I}$, and $\mathbb{K} = -\mathbb{A}$, Eq. 33 becomes

$$\underset{\sim}{y}_{n+1} = (\mathbb{I} - \triangle t\mathbb{A})^{-1} \left(\underset{\sim}{y}_n + \triangle t \underset{\sim}{f}_{n+1} \right), \tag{34}$$

which is identical to Eq. 4. Thus, according to Dahlquist's theorem, this is the implicit scheme for the solution of (1) with the smallest local truncation error possible.

3 Code Implementation

The IMEX_a algorithm is called from the Matlab function IMEX_a.m by

[eta, t, F, UD] = IMEX_a (func, Amat, t0, t1, params).

This function has four outputs, eta, t, F, and UD, which are described in Table 1. If an output quantity is not needed, it may be replaced in the function call by a $\sim$.

Output Quantity	Description
eta	The system's solution, given in Eq. 1 as $\underset{\sim}{y}$, in matrix form with each column corresponding to each output time step.
t	The vector of output time steps.
F	The matrix of forces applied to the system, given in Eq. 1 as $\underset{\sim}{f}$, with each column corresponding to each output time step.
UD	The user defined output structure specified by the options described in Table 3.

Table 1. Description of the outputs of the IMEX_a algorithm.

Only the first four arguments of IMEX_a.m (func, Amat, t0, and t1) are required; the remaining arguments contained within the structure params, if left unspecified, are assigned default values. The required arguments are detailed in Table 2.

Input Argument	Description
func	The force function for the simulation, given in Eq. 1 as $\underset{\sim}{f}$.
Amat	The system matrix for the simulation, given in Eq. 1 as $\mathbb{A}$.
t0	The starting time for the simulation.
t1	The final time for the simulation.

Table 2. Description of required inputs for the IMEX_a algorithm.

With these four arguments, a system described by Eq. 1 is simulated from time t0 to t1. The remaining arguments, which are not required, are summarized in Table 3. The parameters of Table 3 are designed to give the user control over the convergence properties of the simulation (with dt,

dtmin, dtmax, tol, IC, and method), the output resolution and metrics (output_res, output_flag, and UO_func), and the termination criteria (ES_func, cutoff). The two user specified functions in particular (UO_func and ES_func), enable the user to perform such actions as record impact velocities or terminate the simulation after a particular system state is achieved. This functionality is further documented in the next section.

Parameter	Description
dt	The initial time step to use, $\triangle t$ in Eq. 4. After the first time step, $\triangle t$ is automatically adjusted by the algorithm. If unspecified, $\mathsf{dt} = (\mathsf{t1} - \mathsf{t0})/10^6$.
dtmin	The minimum time step size for the simulation. This minimum places a lower bound on $\triangle t$. If unspecified, $\mathsf{dtmin} = \min(\mathsf{dt}, 10^{-12})$.
dtmax	The maximum time step size for the simulation. This maximum places an upper bound on $\triangle t$. If unspecified, $\mathsf{dtmax} = 10^{-3}$.
tol	The tolerance for the error in calculating the adaptive time step, given in Eq. 14 as τ. If unspecified, $\tau = 10^{-3}$.
IC	The vector of initial conditions for the degrees of freedom in $\underset{\sim}{y}$. If unspecified, this is set to be a vector of zeros with the same length as Amat.
output_res	The temporal resolution of the output. If unspecified, the output is returned for 100 approximately evenly spaced time increments spanning $[\mathsf{t0}, \mathsf{t1}]$.
output_flag	Determines whether or not to display progress updates. If $\mathsf{output\_flag} = 1$, progress updates will be displayed in the command window, otherwise they will not be displayed.
UO_func	A user defined output function that specifies quantities to be saved into the UD output structure. By default, this feature is disabled unless specified, and the output variable UD is returned as having a value of 0.
ES_func	A user defined early termination criterion. By default, this feature is disabled unless specified.
cutoff	In case of poor convergence, this is the elapsed cpu time (in seconds) to terminate the simulation at. By default, this is set to approximately 35 days.
method	An option to use either the fifth order accurate Dormand-Prince method (the default, and chosen explicitly by specifying $\mathsf{method} = 5$), the third order accurate Bogacki-Shampine method (chosen by specifying $\mathsf{method} = 3$), the second order accurate Midpoint method ($\mathsf{method} = 2$), or the first order accurate Euler method ($\mathsf{method} = 1$).

Table 3. Description of optional parameters for the IMEX_a algorithm.

The algorithm, itself, uses five Matlab functions: IMEX_a.m, RK5.m, RK3.m, RK2.m, and RK1.m. In IMEX_a.m, the parameters for the simulation are established, and during the simulation, the output quantities described in Table 1 are generated. The implicit portion of the algorithm is

found in IMEX_a.m, and the explicit portion and the adaptive time stepping algorithms exist in the files RK5.m, RK3.m, RK2.m, and RK1.m, depending on whether the Dormand-Prince method (RK5.m), the Bogacki-Shampine method (RK3.m), the Midpoint method (RK2.m), or the Euler method (RK1.m) is specified, all of which are called by IMEX_a.m.

4 Example 1: Computational Savings of the Method Demonstrated on A Two Pawl Mechanism with Impacts

As an example application, the two pawl system illustrated in Fig. 1 is presented. This mechanism consists of two preloaded pawls that are part of a larger latching mechanism. Each pawl has moment of inertia J_j, rotational spring stiffness K_j, mass M_j, preload angle φ_j, equivalent viscous damping c_j, applied torque due to external excitation T_j, angular displacement θ_j, and a center of mass located a distance L_{Gj} from the center of rotation. The first pawl, taken to be on the left, is preloaded against and constrained from displacing in the negative θ_1 direction by a contact pin located at a distance ℓ from the center of rotation of the pawl. The second pawl, conversely, is preloaded against the first pawl. Under certain excitations, it is possible for the second pawl to unintentionally slip under the first pawl, which would be considered a device failure. The two pawls nominally contact one another at a distance L_1 from the center of rotation of the first pawl and a distance L_2 from the center of rotation of the second pawl.

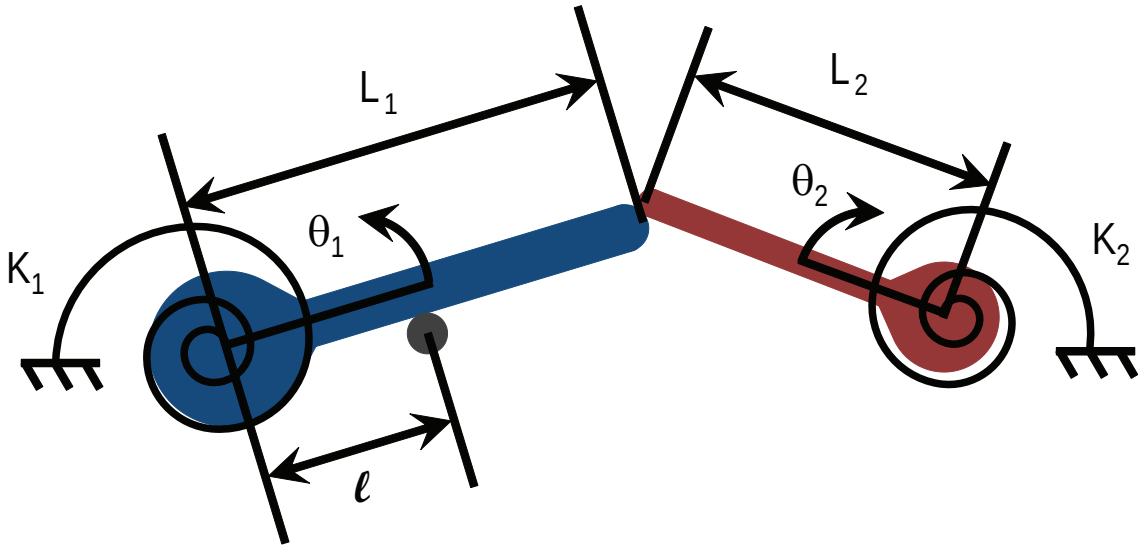

Figure 1. The two pawl system considered in the example application.

The equations of motion for the two pawls are given in state space form as

$$\begin{Bmatrix} \dot{\theta}_1 \\ \ddot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_2 \end{Bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -K_1/J_1 & -c_1/J_1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_2/J_2 & -c_2/J_2 \end{bmatrix} \begin{Bmatrix} \theta_1 \\ \dot{\theta}_1 \\ \theta_2 \\ \dot{\theta}_2 \end{Bmatrix} + \begin{Bmatrix} 0 \\ \frac{F_2\ell - F_1L_1 + T_1 + K_1P_1}{J_1} \\ 0 \\ \frac{F_1L_2 + T_2 + K_2P_2}{J_2} \end{Bmatrix}. \quad (35)$$

The nonlinear forces acting on the pawls are

$$F_1 = F_C(\theta_1 L_1 - \theta_2 L_2, \dot{\theta}_1 L_1 - \dot{\theta}_2 L_2) \tag{36}$$

$$F_2 = F_C(-\theta_1 \ell, -\dot{\theta}_1 \ell), \tag{37}$$

where F_C is specified by the constitutive model used for contact [4] and is a function of contact interference and relative velocity. The excitation T_j is modeled as a haversine impulse acting on the center of mass of the two pawls

$$T_j = \begin{cases} M_j L_{Gj} \Omega \left(1 - \cos\left(\frac{2\pi t}{\tau}\right)\right) & 0 \le t \le \tau \\ 0 & \text{otherwise.} \end{cases} \tag{38}$$

with time t.

Casting this problem into the framework of Eq. 1,

$$\underset{\sim}{y} = \begin{Bmatrix} \theta_1 \\ \dot{\theta}_1 \\ \theta_2 \\ \dot{\theta}_2 \end{Bmatrix} \tag{39}$$

$$\mathbb{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -K_1/J_1 & -c_1/J_1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_2/J_2 & -c_2/J_2 \end{bmatrix} \tag{40}$$

$$\underset{\sim}{f} = \begin{Bmatrix} 0 \\ \frac{F_2\ell - F_1 L_1 + T_1 + K_1 P_1}{J_1} \\ 0 \\ \frac{F_1 L_2 + T_2 + K_2 P_2}{J_2} \end{Bmatrix}. \tag{41}$$

Once these equations are coded into Matlab, IMEX_a.m is called via

```
params.dt = 10^-7;
params.dtmin = 10^-9;
params.dtmax = 10^-3;
params.tol = 10^-6;
params.IC = [0; 0; acos(1 - (r1 + r2)^2/2L2^2); 0];
params.output_res = 10^-5;
params.output_flag = 1;
params.UO_func = @(t,dt,y,UD,flag) Mech_Output_func(t,dt,y,UD,sysparams,flag);
params.ES_func = @(t,y,UD) Mech_ES_func(t,y,UD,sysparams);
params.cutoff = 6000;
params.method = 5;
[eta, t, F, UD] = IMEX_a.m(@(t,y,UD) Mech_force_func(t,y,sysparams,UD),
                           A, 0, 1, params);
```

Each of the functions called above are detailed in what follows. The structure sysparams contains all of the pertinent geometric and material properties for the system.

4.1 An Example Force Function

This subsection provides sample code for @(t,y,UD) Mech_force_func(t,y,sysparams,UD), the force function referenced above.

```
function dY = Mech_force_func(T, Y, sysparams, ~ )
% Function Mech_force_func calculates the force vector for the two pawl mechanism.
% It takes inputs: time T, state vector Y, and geometric/material parameters sysparams
% It returns the force vector dY for the system described by dydt = Ay+f.
% The structure UD is not needed in this function, and is thus replaced with a ~.

% Calculate the torques at this time step due to the Haversine shock
if T < sysparams.Duration
  Tapl1 = sysparams.inertia1*(1-cos(T*(1/sysparams.Dur*2*pi)))/2 ;
  Tapl2 = sysparams.inertia2*(1-cos(T*(1/sysparams.Dur*2*pi)))/2 ;
else
  Tapl1 = 0 ;
  Tapl2 = 0 ;
end

% Calculate the gap between the pawls
gap = sqrt((sysparams.l2*(1-cos(Y(3)))+sysparams.l1*(1-cos(Y(1))))^2 ...
       + (sysparams.l2*sin(Y(3))-sysparams.l1*sin(Y(1)))^2) - sysparams.r1 - sysparams.r2 ;

% Calculate contact forces between the pawls
if gap > 0
  % Reset the contact model, code omitted.
  F1 = 0 ;
else
  F1 = sign(sin(Y(3))*sysparams.l2-sin(Y(1))*sysparams.l1) ...
        *contactForce([-gap; Y(2)*sysparams.l1-Y(4)*sysparams.l2]) ;
end

% Look at the gap between pawl one and the contact pin
if Y(1) > 0
   % Reset the contact model, code omitted.
  F2 = 0 ;
else
  F2 = contactForce([-Y(1)*sysparams.l3; -Y(2)*sysparams.l3]) ;
end

% Cast forces in the vector structure necessary for the algorithm
dY = [0;
     (sysparams.l3*F2-sysparams.l1*F1+Tapl1+sysparams.preload1)/sysparams.J1;
```

```
    0;
    (sysparams.l2*F1+Tapl2+sysparams.preload2)/sysparams.J2] ;

clear Tapl1 Tapl2 F1 F2 gap
end
```

4.2 An Example Output Function

The function @(t,dt,y,UD,flag) Mech_Output_func(t,dt,y,UD,sysparams,flag), which is called above, is the user defined output function, and is detailed in what follows. The variable flag is set by the main function of the algorithm. When flag = 1, the algorithm is recording data that corresponds to the output time vector t in the call to IMEX_a.m. Thus, for an output metric that has the same temporal values as the solution (eta in the original call), data should be recorded only when flag = 1. At all other intermediate time steps (flag = 0), it is recommended that only integral/scalar quantities be calculated, such as wear work rates, as creating a vector of quantities indexed at every intermediate time step potentially could become prohibitively large (that is, on the order of millions of elements).

```
function UD = Mech_Output_func(~,dt,Y,UD,sysparams,flag)
% Function Mech_Output_func calculates the gaps, contact forces, and wear work rates for the
% two pawl mechanism.
% The input for time is not required, and is thus replaced with a ~ in the declaration.
% It takes inputs: time step size dT, state vector Y, and geometric/material parameters
% sysparams, and output flag flag.  It returns the structure UD, which includes a vector of gap
% sizes and contact forces recorded at every time step for debugging purposes, as well as the
% wear work rates recorded at the same resolution as output_res in the main algorithm.

% Calculate the gap between the pawls
gap = sqrt((sysparams.l2*(1-cos(Y(3)))+sysparams.l1*(1-cos(Y(1))))^2 ...
      + (sysparams.l2*sin(Y(3))-sysparams.l1*sin(Y(1)))^2) - sysparams.r1 - sysparams.r2 ;

% Calculate contact forces between the pawls
if gap > 0
 f1 = 0 ;
else
 f1 = sign(sin(Y(3))*sysparams.l2-sin(Y(1))*sysparams.l1) ...
     *contactForce([-gap; Y(2)*sysparams.l1-Y(4)*sysparams.l2]) ;
end

% Calculate contact forces between pawl one and the contact pin
if Y(1) > 0
 f2 = 0 ;
else
```

```
  f2 = contactForce([-Y(1)*sysparams.l3; -Y(2)*sysparams.l3]) ;
end

% For flag == 1, data is being archived by the main algorithm for this time step.
% However, it may be useful to record additional, data on the forces and gaps.
% UD.gaps, UD.f1, and UD.f2 will all have the same length as the time vector
% returned by the main algorithm.
if flag == 1
  if isfield(UD,'gaps')
    UD.gaps(end+1) = gap ;
    UD.f1(end+1) = f1 ;
    UD.f2(end+1) = f2 ;
  else
    UD.gaps = gap ;
    UD.f1 = f1 ;
    UD.f2 = f2 ;
  end

% For flag == 0, no data is being archived by the main algorithm, so quantities of
% interest must be recorded, specifically the wear work rates for the system:
else
  if isfield(UD,'wears')
    UD.wears = UD.wears + ...
            dt*[abs(f1*(Y(2)*sysparams.l1-Y(4)*sysparams.l2));
               abs(f2*(Y(2)*sysparams.l3))] ;
  else
    UD.wears = dt*[abs(f1*(Y(2)*sysparams.l1-Y(4)*sysparams.l2));
               abs(f2*(Y(2)*sysparams.l3))] ;
  end
end

clear gap f1 f2
end
```

4.3 An Example Early Termination Function

The last user defined function @(t,y,UD) Mech_ES_func(t,y,UD,sysparams), is the early termination function called above.

```
function S_flag = Mech_ES_func(T,Y,~,sysparams)
% Function Mech_ES_func determines whether to terminate the simulation based off of
% the displacement of the pawls.
% It takes inputs: time T, state vector Y, and geometric/material parameters sysparams.
```

```
% For this application, the input UD is not needed and is replaced with a ~.
% It returns the early termination flag S_flag.
% If S_flag = 0, continue, else terminate the simulation.
% Early termination is defined here as either the second pawl slipping below the first,
% or the first pawl returning to its initial position after the end of the shock.

S_flag = ((T > sysparams.Duration) && (Y(1) < 0 || Y(3) < 0)) ;
end
```

4.4 Comparison to Other Integration Methods

Property	Value
Pawl 1	
Damping Coefficient, c_1	10 N·mm·s/rad
Moment of Inertia, J_1	0.32 N·mm^2
Spring Stiffness, K_1	46.5 N·mm/rad
Length, L_1	12 mm
Center of Mass, L_{G1}	1.6 mm
Contact Pin Distance, ℓ	4 mm
Mass, M_1	1.5 g
Preload Angle, φ_1	1 rad
Pawl 2	
Damping Coefficient, c_1	1 N·mm·s/rad
Moment of Inertia, J_2	0.01 N·mm^2
Spring Stiffness, K_2	8.8 N·mm/rad
Length, L_2	5 mm
Center of Mass, L_{G2}	0.4 mm
Mass, M_2	0.2 g
Preload Angle, φ_2	2.9 rad
Excitation	
Nominal Shock Amplitude, Ω	100,000 g's
Nominal Shock Duration, τ	1 ms

Table 4. Properties of the two pawl mechanism.

The performance of the IMEX_a algorithm for simulating the dynamics of the two pawl system of Fig. 1 with properties listed in Table 4 is compared to that of six other algorithms that are built into Matlab and described in Table 5. Of note, ode23t uses a method based on the generalized trapezoidal rule, which is the same as the implicit step for the IMEX_a algorithm. As well, ode45 uses a Dormand-Prince fifth order Runge-Kutta adaptive time stepping method, which is the same

method used for the default explicit step of the IMEX_a algorithm, and ode113 uses a Bogacki-Shampine third order Runge-Kutta adaptive time stepping method, which is the same method used for the explicit step of the IMEX_a3 algorithm. The ode23tb algorithm is a multi-step method with a low order Runge-Kutta first step followed by a backward Euler second step, which is similar to the basic IMEX algorithm detailed in Section 2.1. Two of the remaining algorithms are implicit methods (ode23t and ode15s), and the last three algorithms are explicit methods (ode23s, ode45, and ode113).

Method	Description
ode15s	A multi-step, medium accuracy solver for stiff systems that uses implicit backward differentiation formulas.
ode23s	A one-step, low accuracy solver for stiff systems that uses the second order Rosenbrock formula.
ode23t	A one-step, low accuracy solver for moderately stiff systems based on the trapezoidal rule.
ode23tb	A multi-step, low accuracy solver for stiff systems that uses an implicit Runge-Kutta formula with a backward differentiation second step.
ode45	A one-step, medium accuracy solver for nonstiff systems that uses a fifth order Runge-Kutta (Dormand-Prince) method.
ode113	A multi-step, medium accuracy solver for nonstiff systems that uses a third order Runge-Kutta (Bogacki-Shampine) method.

Table 5. Built in solvers for differential equations in Matlab.

The computational efficiency of the seven different algorithms (including the default Dormand-Prince fifth order adaptive time step algorithm referred to as IMEX_a and the Bogacki-Shampine third order adaptive time step algorithm referred to as IMEX_a3) is detailed in Table 6. The mean values and standard deviations listed in Table 6 are calculated based off of ten different simulations using each method. The IMEX_a method is demonstrated to be the most computationally efficient of the numerical methods studied, with the third order explicit step implementation of IMEX_a3 requiring only 5% more time for this specific problem due to the stiffness of the system. The next most efficient method is, unsurprisingly, ode23t (which is based on the generalized trapezoidal rule). The Dormand-Prince method (ode45) is observed to be the least computationally efficient approach for this problem, as it is not well suited for simulating the dynamics of stiff systems.

Figure 2 shows the predicted response for five of these algorithms for the same input. The two methods not shown (ode15s and ode23s) show similar trends. Somewhat alarmingly, none of the algorithms agree despite the plot showing converged results for each. The generalized trapezoidal method (ode23t) incorrectly predicts that the mechanism will fail (i.e. the right pawl slips under the left pawl as is evidenced by a negative gap), while all other algorithms predict that the right pawl never slips under the left pawl. Thus, even though ode23t is the next fastest method, the accuracy is too low to be considered useful for this problem. Similarly, the other implicit method, ode15s, also predicts incorrect behavior: namely oscillations about an unstable point in the system. Clearly,

Method	Mean Computational Time (s)	Standard Deviation (s)	Increase Over IMEX_a
IMEX_a	0.343	0.0023	—
IMEX_a3	0.361	0.0027	5%
ode15s	0.495	0.0111	44%
ode23s	2.232	0.0485	552%
ode23t	0.411	0.0717	20%
ode23tb	1.317	0.0706	284%
ode45	2.368	0.0323	591%
ode113	2.282	0.0387	566%

Table 6. Computational efficiencies of the algorithms for the two pawl mechanism.

for the nonlinearities involved in this system, the two implicit methods built into Matlab are not suitable. The numerical damping introduced by the explicit methods, such as ode45, is evident in the increased settling time following the initial transient (i.e. the longer predicted settled gap size). The most similar method, ode23tb, which uses a two step, implicit-explicit scheme, yields similar results but requires an additional 284% of the computational time required by IMEX_a.

When the computational performance is extrapolated to the number of simulations required for the robust optimization of this system, which could require over 100,000 simulations, the computational savings become more dramatic. For a robust optimization that has 100,000 simulations, the IMEX_a method will require 9.5 hours of computational time. The next most efficient methods, the two fully implicit methods (ode15s and ode23t), would require 13.75 and 11.5 hours of computational time respectively; however, these two methods demonstrated the lowest amount of accuracy as noted above. By contrast, the explicit methods each have reasonable predictions for the dynamics, though they introduce significant amounts of numerical damping. In terms of computational efficiency, though, they require 60-66 hours of computational time. For models that require significantly more computational time, say 10 seconds for one simulation using the IMEX_a method, the computational savings for a robust optimization requiring 100,000 simulations becomes even more significant: 11.5 days for the IMEX_a method compared to 75 to 80 days for the explicit methods. Thus, for this type of problem, the IMEX_a algorithm proves to be the best choice due to the computational efficiency, lack of numerical damping introduced into the system, and reasonable predictions of the system's dynamics.

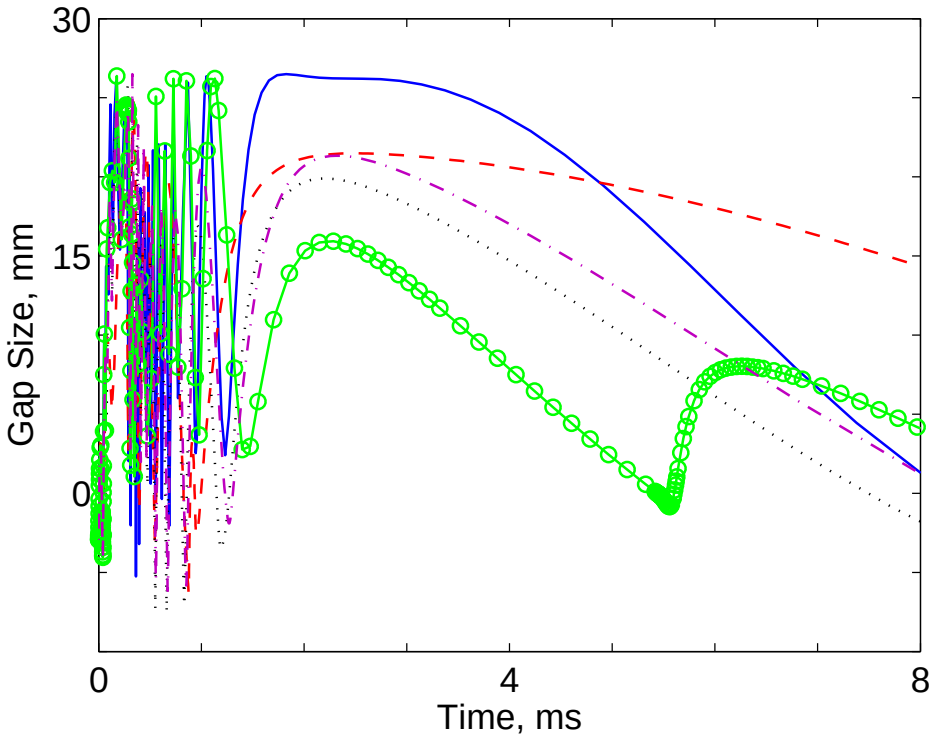

Figure 2. The gap size in the two pawl system following a shock. Shown are the responses predicted by IMEX_a (blue, —), ode45 (red, – –), ode23t (black, · · ·), ode23tb (purple, – · –), and ode113 (green, ○).

5 Example 2: Accuracy of the Method Demonstrated on the Simulation of an Acoustic Standing Wave

In the previous example, the computational efficiency of the IMEX_a method is discussed; however, due to the lack of a closed form solution for the response of the system, it is difficult to assess the relative accuracy of each method presented. In this second example, a known solution exists such that the accuracy of the IMEX_a method can be calculated. An acoustic standing wave is modeled using the one-dimensional linearized Euler equations [16], which can be expressed in the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\left\{\begin{array}{c} u \\ p \end{array}\right\} = \mathbb{A}\left\{\begin{array}{c} u \\ p \end{array}\right\} \tag{42}$$

in terms of the velocity field u and pressure field p. For details of the modeling of this system or the exact, analytical solution, refer to [16]. Despite the straightforward form of Eq. 42, numerical solution of this system is often challenging due to the added damping and period error inherent in most numerical schemes [17].

Because the system is expressed without a force vector, there are two options for how the IMEX_a method can be called. The first option is to consider there to be no force function. In this framework, calls to the IMEX_a method will result in a purely implicit integration scheme that is based on the generalized trapezoidal method (and is referred to as IMEX_a Implicit in the following discussion):

$$[\mathsf{eta},\, \mathsf{t},\, \sim,\, \sim] = \mathsf{IMEX\_a}\,(@(\mathsf{t},\mathsf{y},\mathsf{UD})\ 0,\ \mathbb{A},\ 0,\ 100);$$

where the simulation spans a time from 0 to 100 s. The alternative approach, which is adopted in what follows, is to treat the right hand side of Eq. 42 as a force term. This approach results in a purely explicit integration scheme:

$$[\mathsf{eta},\, \mathsf{t},\, \sim,\, \sim] = \mathsf{IMEX\_a}\,(@(\mathsf{t},\mathsf{y},\mathsf{UD})\ \mathbb{A}\times\mathsf{y},\ \mathrm{zeros}(\mathrm{length}(\mathbb{A})),\ 0,\ 100);$$

Figure 3 presents the relative errors of the IMEX_a method and the ode45 method normalized with respect to the norm of the exact solution. The mean of the error for ode45 is 4.4% for the velocity field, and 5.7% for the pressure field, whereas the mean of the error for the IMEX_a method is 3.1% for the velocity field and 2.7% for the pressure field. Thus, the error for the velocity field and pressure field is 42% and 111% less, respectively, for the IMEX_a method as compared to the ode45 algorithm. At the same time, the IMEX_a required 92% of the computational time that ode45 required. For more coarse tolerances on both methods, significantly faster solutions can be achieved; however, the errors significantly increase. Comparisons of the relatively accuracy and efficiency of several different methods are presented in Table 7.

All results presented in Table 7 are calculated using the same error tolerance and maximum step size (for both the IMEX_a toolbox and the ode family of algorithms, with the third order Bogacki-Shampine method given as IMEX_a3, the second order Midpoint method given as IMEX_a2, and

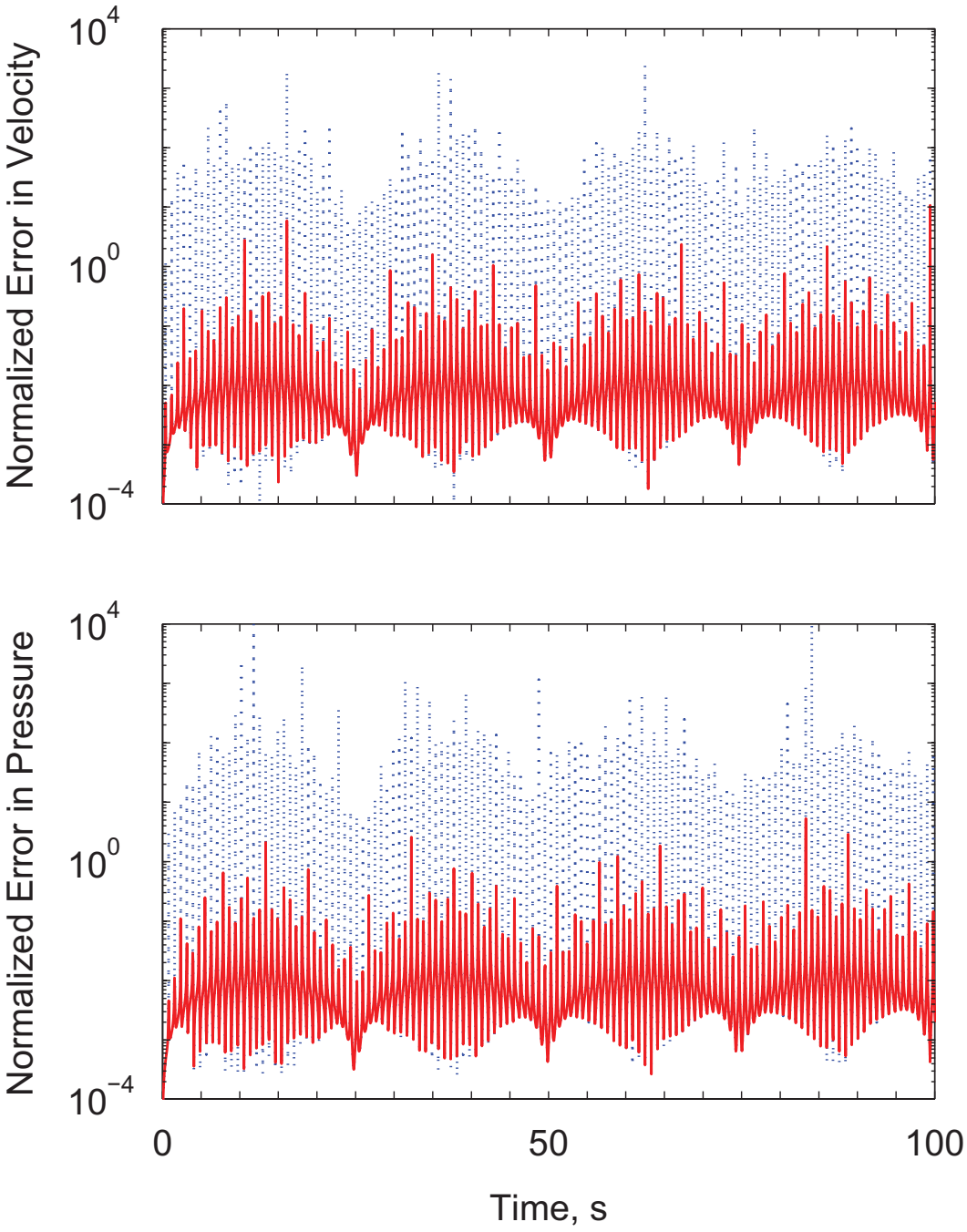

Figure 3. The relative error of the numerical solutions of the acoustic standing wave in one-dimension (normalized with respect to the norm of the exact solution) for the IMEX_a method (red, —), and the ode45 method (blue, $\cdots$).

Method	Difference in Computational Time From IMEX_a	Normalized Error in Velocity	Normalized Error in Pressure
IMEX_a	—	3.0%	2.6%
IMEX_a3	-18.9%	3.0%	2.6%
IMEX_a2	+886.1%	3.8%	3.0%
IMEX_a1	+2602.8%	38.4%	171.6%
IMEX_a Implicit	+35.4%	164.0%	151.8%
ode15s	+3.2%	4.3%	5.3%
ode23s	+2680.3%	4.2%	4.6%
ode23t	-41.0%	4.5%	4.6%
ode23tb	-6.1%	4.2%	4.6%
ode45	+8.2%	4.4%	5.7%
ode113	-46.7%	4.2%	4.9%

Table 7. Computational efficiencies and accuracies of the algorithms for the acoustic standing wave.

the first order Euler method given as IMEX_a1). Because this example is a purely linear problem, it does not take advantage of the numerical efficiencies that the IMEX_a algorithm demonstrates for stiff, nonlinear systems; however, the performance of the IMEX_a algorithm is still favorably compared to the other methods. From the results of Table 7, several conclusions are able to be made. First, casting the problem such that a purely implicit solution results from running the IMEX_a toolbox (listed as IMEX_a Implicit) results in an inaccurate and inefficient solution due to the fixed step size and no application of error control. Second, with the maximum step sizes and error tolerance held constant for all algorithms, the lowest error, as compared to the exact solution, is achieved using both IMEX_a and IMEX_a3. Other methods, for this specific problem, exhibit low errors and fast computational times (such as ode23t, ode23tb, and ode113), but the percent error introduced by these methods is between 40% and 80% greater than the error observed with the IMEX_a method. Increasing error tolerances for the other methods can result in a significantly improved solution; however, the computational time simultaneously increases, and the error is not reduced to a level below that exhibited by the IMEX_a method. This demonstrates that not only is the IMEX_a method more efficient than other available numerical integration schemes (when applied to stiff, nonlinear problems), but it is also more accurate than the other methods considered for this problem.

6 Summary

The derivation of the IMEX_a ordinary differential equation numerical integration algorithm is presented in this report. The algorithm consists of two steps: an explicit predictor step, and an implicit solution step. The explicit step is based on a fifth order Runge-Kutta explicit step known as the Dormand-Prince algorithm, which adaptively modifies the time step by calculating the error relative to a fourth order estimation. The implicit step is based on a backward Euler method, and is shown via Dahlquist's theorem to minimize error in the system. An alternative explicit step, the third order Runge-Kutta method attributed to Bogacki and Shampine, is also included in the toolbox for problems with less stringent accuracy tolerances or less stiff systems in addition to a first order Euler method and a second order Midpoint method for well behaved linear systems.

Sample code is presented in order to demonstrate how the IMEX_a algorithm can be incorporated into a simulation framework, along with examples of the optional functions used by the IMEX_a algorithm. The code is developed to give the user a large amount of control over the convergence and termination criteria of the algorithm, as well as specifying complex, user defined output metrics.

Finally, this report presents two examples. The first example, a two pawl mechanism, is presented in order to quantify the computational advantages of the IMEX_a algorithm over six other widely used numerical solvers when applied to a stiff, nonlinear system. It is demonstrated that the IMEX_a algorithm is more computationally efficient than each of the other solvers. Further, in comparing the IMEX_a algorithm to the numerical solvers that made reasonable predictions of the system's dynamics, the IMEX_a algorithm required approximately 12% of the computational time required by the other algorithms, a significant computational savings that is expected to further enable sensitivity and optimization studies. The second example, a one-dimensional acoustic standing wave, is presented to demonstrate that the IMEX_a algorithm can be more accurate than other numerical algorithms. For this particular example, the IMEX_a algorithm is demonstrated to have 40% and 80% less error than the ode family of solvers.

References

[1] M. R. Brake, Analytical Modeling of the MC4713 Launch Accelerometer Dynamics, 5283947-PR (OUO). Sandia National Laboratories, Albuquerque, NM, 2010.

[2] M. R. Brake, J. E. Massad, R. C. Smith, B. Beheshti, K. Chowdhary, J. Davis, S. Wang, Uncertainty enabled design of an acceleration switch, in: Proceedings of ASME 2011 International Mechanical Engineering Congress and Exposition, Denver, CO, 2011.

[3] M. R. Brake, An analytical elastic-perfectly plastic contact model, *International Journal of Solids and Structures* 49 (2012) 3129–3141.

[4] M. R. Brake, The effect of the contact model on the impact-vibration response of continuous and discrete systems, *Journal of Sound and Vibration* 332 (2013) 3849–3878.

[5] M. R. Brake, H. Sumali, D. S. Aragon, P. L. Reu, M. V. Bejarano, D. J. VanGoethem, Development and validation of an elastic-perfectly plastic contact model for rigid body dynamics simulations, in: European Congress on Computational Methods in Applied Sciences and Engineering, Vienna, Austria, 2012.

[6] M. R. Brake, D. S. Aragon, D. J. VanGoethem, H. Sumali, The effect of contact model on the design of mechanical systems, in: ASME 2012 International Mechanical Engineering Congress and Exposition, Houston, TX, 2012.

[7] M. R. Brake, D. J. Segalman, A new approach to modeling discrete nonlinear constraints in continuous systems: The method of discontinuous basis functions, in: Proceedings of ASME 2011 International Design Engineering Technical Conference and Computers and Information in Engineering Conference, Washington, DC, 2011.

[8] M. R. Brake, D. A. Hills, Determination of the limits of quasi-static and dynamic solutions for problems with frictional interfaces, in: 7th International Symposium on Fretting Fatigue, Oxford, UK, 2013.

[9] M. R. Brake, D. J. Segalman, Nonlinear model reduction of von Karman plates under quasi-steady fluid flow, *AIAA Journal* 48 (2010) 2339–2347.

[10] M. R. Brake, M. F. Barone, D. J. Segalman, Nonlinear model reduction of von Karman plates under linearized compressible fluid flow, *AIAA Journal* 50 (2012) 1047–1059.

[11] M. R. Brake, A hybrid approach for the modal analysis of continuous systems with nonlinear constraints, *Journal of Sound and Vibration* 330 (2011) 3196–3221.

[12] M. R. Brake, Chatter Analysis of Flexible Spring Contacts in an Assembly, 5305328-PR (OUO). Sandia National Laboratories, Albuquerque, NM, 2012.

[13] T. J. R. Hughes, The Finite Element Method; Linear Static and Dynamic Finite Element Analysis, Dover Publications, Inc., 2000.

[14] J. R. Dormand, P. J. Prince, A family of embedded Runge-Kutta formulae, *Journal of Computational and Applied Mathematics* 6 (1980) 19–26.

[15] E. Hairer, S. P. Nørsett, G. Wanner, Solving Ordinary Differential Equations I: Nonstiff Problems, Springer-Verlag, 1993.

[16] M. F. Barone, I. Kalashnikova, D. J. Segalman, H. K. Thornquist, Stable Galerkin reduced order models for linearized compressible flow, *Journal of Computational Physics* 228 (2009) 1932–1946.

[17] C. I. Morris, Studies of inviscid flux schemes for acoustics and turbulence problems, in: 51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, Grapevine (Dallas/Ft. Worth Region), Texas, 2013.

DISTRIBUTION:

1	MS 1070	Matthew R. Brake, 1526 (electronic)
1	MS 0346	Brenton Elisberg, 1526 (electronic)
1	MS 0346	Dannelle S. Aragon, 1526 (electronic)
1	MS 0346	Diane Peebles, 1526 (electronic)
1	MS 0346	Kurtis R. Ford, 1526 (electronic)
1	MS 0346	Ryan D. Jamison, 1526 (electronic)
1	MS 0349	Gilbert L. Benavides, 2613 (electronic)
1	MS 0350	Catherine M. Siefert, 2613 (electronic)
1	MS 0350	Marcus J. Craig, 2613 (electronic)
1	MS 0351	Charles Dennis Croessmann, 1520 (electronic)
1	MS 0372	Douglas VanGoethem, 1526 (electronic)
1	MS 0825	Matthew F. Barone, 1515 (electronic)
1	MS 1064	Matthew D. Williams, 2615 (electronic)
1	MS 1070	Jordan E. Massad, 1526 (electronic)
1	MS 1320	Irina Kalashnikova, 1442 (electronic)
1	MS 9042	Daniel J. Segalman, 8259 (electronic)
1	MS 0899	Technical Library, 9536 (electronic copy)

Sandia National Laboratories