EXTENDING EXPLICIT AND LINEARLY IMPLICIT ODE SOLVERS FOR INDEX-1 DAES

Matthew T. Lawder[1], Venkatasailanathan Ramadesigan[2], Bharatkumar Suthar[1], and Venkat R. Subramanian[3]

1-Washington University in St. Louis, 1 Brookings Drive, St. Louis, MO 63130
mtlawder@wustl.edu
b.suthar@wustl.edu

2-Indian Institute of Technology-Bombay, Powai, Mumbai, Maharashtra 400076, India
venkatr@iitb.ac.in

3-University of Washington, Seattle, 105 Benson Hall, Seattle, WA 98195
Pacific Northwest National Laboratory, 902 Battelle Blvd., Richland, WA 99354
vsubram@uw.edu, corresponding author

ABSTRACT

Nonlinear differential-algebraic equations (DAE) are typically solved using implicit stiff solvers based on backward difference formula or RADAU formula, requiring a Newton-Raphson approach for the nonlinear equations or using Rosenbrock methods specifically designed for DAEs. Consistent initial conditions are essential for determining numeric solutions for systems of DAEs. Very few systems of DAEs can be solved using explicit ODE solvers. This paper applies a single-step approach to system initialization and simulation allowing for systems of DAEs to be solved using explicit (and linearly implicit) ODE solvers without *a priori* knowledge of the exact initial conditions for the algebraic variables. Along with using a combined process for initialization and simulation, many physical systems represented through large systems of DAEs can be solved in a more robust and efficient manner without the need for nonlinear solvers. The proposed approach extends the usability of explicit ODE solvers and removes the requirement of Newton-Raphson type iteration.

1. INTRODUCTION

When modeling physical phenomena, the representative system of equations often includes a combination of ordinary differential equations (ODEs), partial differential equations (PDEs), and algebraic equations (AEs). These phenomena often use PDEs as governing equations of the system that vary in both space and time. When discretizing these PDEs spatially, the system becomes a set of ODEs and AEs. The resulting set of equations is known as a system of differential algebraic equations (DAEs). In physical systems, ODEs will typically represent most of the governing equations, while AEs act as constraints applied to the system ensuring that the solution accurately reflects the physical possibilities (e.g. conservation laws, boundary conditions, etc.). Systems of DAEs have been used to model real-world environments across a range of problems including large industrial processes (Pantelides et al., 1988, Li et al., 2012), predator-prey eco-systems (Li and Petzold, 2000), electric power systems (Praprost and Loparo, 1996, Susuki et al., 2008), and electrochemical environments. Computing solutions for systems of DAEs is important across a variety of fields. These systems combine algebraic states that act as constraints (nonlinear) on a set of ODEs, making the system more difficult to solve (Shampine et al., 1999).

Many different solvers are available for computing systems of DAEs (Hindmarsh, 1980, Petzold, 1982, Berzins et al., 1989, VanKeken et al., 1995, Hairer and Wanner, 1996, de Swart et al., 1998, Wolfram, 2014, Maplesoft, 2015, Mathworks, 2015). More information on the method and solution procedure of several ODE/DAE solvers, are provided elsewhere (Cash, 2003, Cellier and Kofman, 2006, Methekar et al., 2011). Solvers based on backward difference formula (DASKR/DASSL/GEAR) (Brown et al., 1994, Brown et al., 1998) or implicit Runge-Kutta methods (RADAU IIA) (Hairer and Wanner, 1999) are efficient in solving nonlinear DAEs. Other approaches include semi-implicit methods (such as the BESIRK solver (Schwalbe et al., 1996) based on semi-implicit RK) and Rosenbrock methods (Michelsen, 1977, Taylor, 1999). However, semi-implicit methods still require consistent initial conditions (ICs) for algebraic variables. As of today, Maple™, (Maplesoft, 2015) a symbolic programming language, solves nonlinear DAEs by either eliminating algebraic variables (when possible)

or by differentiating the algebraic constraints and inverting the overall system to arrive at an explicit ODE of the form $\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(\boldsymbol{y})$. In addition, many system solvers like DYMOLA (Dassault Systems, 2015) convert DAEs to ODEs by differentiation and then solve the resulting ODE system using DASSL (Petzold, 1982, Cellier and Kofman, 2006), a solver that uses Newton-Raphson type iteration at each time step. In this paper, we show how explicit ODE solvers (such as Maple's rkf45 (Maplesoft, 2015), a fourth-fifth order Runge-Kutta solver) can be used to solve nonlinear DAEs starting from inconsistent algebraic states. In addition, for stiff DAE systems, the method proposed in this paper is applied to Rosenbrock stiff solver typically used for stiff ODEs.

For a system of DAEs, a set of consistent ICs must be given in order to solve the system with standard solvers. Some solvers contain initialization routines that help to calculate consistent ICs from starting guesses. These routines add computational time and often require the additional use of solvers to obtain the ICs used by the primary DAE solver. By having a system that includes both ODEs and AEs, not all ICs offer a possible solution and inconsistent ICs will cause solvers to fail in many instances. Knowledge of the governing equation and the underlying physics of the system can help in choosing consistent ICs.

In many cases though, consistent ICs for all of the variables are not known *a priori* and the effects of operating conditions and system parameters like rate constants or diffusion coefficients may also be unknown. Even small deviations from consistent ICs will cause the solver to fail (using standard solvers) (Methekar et al., 2011). Leimkuhler et al. (1991) outlines the problem of obtaining consistent initial conditions for DAEs and shows examples of different initialization techniques.

The most basic IC estimates use non-physical approximations such as setting differential variable gradients to zero initially, creating solver inefficiencies for the few cases they may be able to solve (Leimkuhler et al., 1991). An approach where the ICs of differential and algebraic variables were calculated separately was proposed by Dew and Walsh (1981). Petzold (1982) outlined a method using an Euler backward step with a very small step size in order to obtain values very near the initial time.

Berzins et al. (1989) combined both methods and expanded the Euler steps to try to provide a more flexible method. Lamour and Mazzia (2009) outlined a method that approximates the system's derivatives and can be used for systems of DAEs up to index-3. However, the method has some restrictions caused by the inner differentiations required for higher indexed systems. An optimization approach for finding ICs through successive linear programming has been proposed by Gopal and Biegler (1998). While studies on general DAE solution methods, especially for higher order systems, remains an active area of research, some methods have focused on obtaining solutions for more specific cases rather than general sets of DAEs. Campbell proposed a Taylor series approximation for linear time-varying cases (Leimkuhler et al., 1991). Other methods have focused on higher order DAEs. A Laplace transform method approach has been used to find initial values by Reissig et al. (2002). A perturbation approach was applied over all variables by Garcia to solve index-3 DAE systems (Garcia, 2000). This approach was modified by Methekar et al. (2011) to use a perturbation only on the required algebraic variables allowing for faster initialization of index-1 DAEs.

The approach outlined in this paper builds upon the two-step process of 1) Perturbation initialization as outlined in Methekar et al. (2011) and 2) DAE simulation based on the consistent ICs obtained from step 1 using explicit (or linearly implicit) ODE solvers. The proposed approach combines the two steps of initialization and by using a switch function (hyperbolic tangent function) to constrain the differential variables during the time when the perturbation approach finds consistent ICs for the algebraic variables. This approach allows the initialization and simulation to be done continuously with a single solver. The proposed single-step approach increases the robustness of the solution method by allowing for larger perturbation values to be applied and increases the computational speed of the solvers while enabling explicit ODE solvers to solve nonlinear DAEs. The proposed approach does not require the use of nonlinear solvers which are common for many initialization subroutines used to find consistent ICs for algebraic variables that arise from nonlinear AEs. The method for applying a perturbation to the algebraic variables and the switch function to the differential variables is shown in the next section and examples using the method are outlined in the following sections. Included are examples outlining stiff

electrochemical systems and an example is included of how the proposed approach can be used to solve implicit ODE systems after converting them into DAE form. Additionally, Appendix A includes the code for solving Example 1 for three different languages: Maple, MATLAB®, and FORTRAN.

2. METHOD

A general DAE system (shown in semi-explicit form) is considered

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(t, \boldsymbol{y}, \boldsymbol{z}) \tag{2.1}$$

$$0 = \boldsymbol{g}(t, \boldsymbol{y}, \boldsymbol{z}) \tag{2.2}$$

where $\boldsymbol{y}$ are the differential variables and $\boldsymbol{z}$ are the algebraic variables. Function $\boldsymbol{g}$ is differentiable and $d\boldsymbol{g}/dt$ is non-singular as the model considered is an index-1 DAE. The system of DAEs shown above often arises from combining equations governing physical phenomena with constraints or discretizing a PDE's spatial variables (while keeping time continuous as shown in Examples 4 and 5). In order to solve the system of DAEs, ICs for all the variables must be given as

$$t = 0; \quad \boldsymbol{y}(0) = \boldsymbol{y}_0; \quad \boldsymbol{z}(0) = \boldsymbol{z}_0 \tag{2.3}$$

However, exact values for consistent $\boldsymbol{z}_0$ are not always readily available. Under normal DAE solvers (without initialization routines), ICs must be consistent with the system of DAEs or a solution cannot be obtained. Variables present in the AEs are limited to sets that directly satisfy the algebraic limits. A system of only ODEs will often offer a wider range of consistent ICs because the equations govern the derivatives (change) of the system rather than the variable values. Combining AEs to an ODE system increases the stiffness of the system and necessitates *a priori* knowledge of the exact ICs of the system. In order to loosen the restriction of consistency on the algebraic variables, a perturbation approach is used as outlined in Methekar et al. (2011) A perturbation parameter, ε, is introduced such that

$$\boldsymbol{g}(t) = \lim_{\varepsilon \to 0} \boldsymbol{g}(t + \varepsilon) = 0 \tag{2.4}$$

and when $t=t+\varepsilon$, the AEs represented in $\boldsymbol{g}$ can be shown as

$$\boldsymbol{g}(t + \varepsilon) = 0 \tag{2.5}$$

Using a Taylor series expansion, we rearrange the AEs to the form

$$-\varepsilon \frac{d\boldsymbol{g}(t)}{dt} = \boldsymbol{g}(t) + O(\varepsilon^2) \tag{2.6}$$

More details on the perturbation approach can be found in Methekar et al. (2011) Once in perturbed form, the AEs can be solved first (without any ODEs, but using the given ICs for the differential variables) to find consistent initial values for all algebraic variables. The algebraic variable values found from the perturbation approach will be consistent with the given differential variable ICs and the values can be used with the initial system of DAEs as ICs. When solving the system of DAEs, the set is solved in its initial form (in the example above, semi-explicit form), and the consistent ICs from initialization are provided. Using this approach provides an initialization routine that produces consistent ICs to be fed into the solver along with the original system. The initialization routine allows for a wider range of initial guesses to be used for the algebraic variables of the system. The perturbation value applied to the system must be small enough so that the converged value of the algebraic variables will be consistent. Larger perturbation values may not converge to consistent ICs. In addition, Maple cannot as of today solve systems of DAEs without first converting the system to ODEs, so even having consistent ICs for algebraic variables won't help in solving nonlinear DAEs.

The above procedure involves two steps: 1) initialization using a solver for obtaining the consistent ICs of the algebraic variables, and 2) the solution of the DAE system calling an additional solver and using the consistent ICs obtained from step 1. Instead of taking a two-step approach to initialization of algebraic variables and solving the complete system, in the proposed approach, a switch function is proposed and used to hold the differential variables static (constant) while the algebraic variables find consistent values using a perturbation approach and then the differential variables are unmasked and the system simulation begins in a continuous manner using the same solver. This single-step approach combines the initialization of the algebraic variables into the solution of the system, offering more robust solutions while reducing system stiffness by relaxing the requirement of the perturbation value. Note that in the proposed approach, the algebraic equations are converted to a perturbed ODE system and therefore

many explicit solvers can be directly used (for example, Maple can handle the perturbed system as an ODE much better than the original DAE). In addition, Maple provides an option to call the ODE solver in compiled form which can be done in a single step minimizing the RAM requirement and the avoiding second calls to dsolve (Maple's inbuilt ODE solver). When running optimal control based on control vector parameterization or estimating parameters from experimental data, it is convenient and efficient to call a single dsolve procedure as opposed to two separate functional calls. Same behavior is observed in MATLAB as well.

The hyperbolic tangent (switch) function applied to the ODEs is formulated as

$$T_H = \frac{1}{2}\left(1+\tanh\left(q(t-t_j)\right)\right) \tag{2.7}$$

where q is a weighting factor determining the discreteness of the function, and t_j is the time allowed for the perturbation approach to find consistent algebraic ICs. The value of t_j will need to be scaled depending on the value of ε used for the perturbation. Subtracting t_j from the total solution time provides the original (simulation) time variable. The switch function is applied to the ODEs as

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}T_H \tag{2.8}$$

The switch function allows for the derivative of the differential variables to be set to zero for the duration of the initialization of the algebraic variables and be set to the function $\boldsymbol{f}$ for the simulation after the initialization period. The adaptive solvers used on the system will find time steps as needed from time $t = 0$ to time = t_j and will find the consistent IC for $\boldsymbol{z}$. The ability of the switch function to approximate a discrete jump with a continuous function makes this approach useful for standard solvers. The representation of discrete events as continuous functions has been shown to be effective in simulation and the switch function approach has even been used to approximate discontinuities as continuous functions (Boovaragavan et al., 2010). The continuous nature of the switch function also allows for minor corrections of the converged ICs at the end of the initialization time (beginning of the actual simulation time). This correction of the converged ICs allow for less restrictive perturbation values to be used in this

approach when compared to the two-step approach. In our experience, the proposed approach also saves a considerable amount of time taken in stopping the solver after initialization, substituting the consistent initial values, and starting the ODE solver again for simulating the entire system of DAEs. Additionally, this method can reduce the time required to properly format the set of equations for solving. With this method, the system of DAEs (Eqs. (2.1) & (2.2)) can be restructured as a single-step ODE system shown by:

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f} T_H \tag{2.9}$$

$$\varepsilon \frac{d\boldsymbol{g}}{dt} = -\boldsymbol{g} \tag{2.10}$$

A schematic qualitatively describing the stages of the single-step approach is shown in Figure 1. Often the perturbed form of the AE (Eq. (2.10)) remains in implicit form because the left hand side will contain differential variables, $\boldsymbol{y}$, or their derivatives, $\frac{d\boldsymbol{y}}{dt}$. This form is acceptable for small systems, but may cause solvers to fail for large sets. In particular, Maple converts all the ODEs of the form $\boldsymbol{M}\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}$ to $\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}$ for all of its solver options if the implicit=true option is not mentioned. Only when $\boldsymbol{M}$ is a constant, the stiff=true option in Maple can be used. The stiff=true option is available for both explicit and linearly implicit ODE solvers based on Rosenbrock methods in Maple for ODEs only when $\boldsymbol{M}$ is a constant. In order to make the system explicit or linearly implicit, first the derivatives of the differential variable must be removed by substituting the original ODE equations (Eq. (2.1)). Then ICs are substituted for the differential variables that remain. By converting to this form, the single-step process will allow for explicit solvers to be used for both the initialization and solution of the system of DAEs.

One can write an equation for the explicit form as

$$\varepsilon \left.\frac{d\boldsymbol{g}}{dt}\right|_{y=y_0, z=z_0, \frac{dy}{dt}=f_0} = -\boldsymbol{g} \tag{2.11}$$

Where Eq. (2.11) can replace Eq. (2.10) for the single-step solution.

Note that Eqs. (2.9), (2.10), & (2.11) are purely ODEs and can be solved using explicit solvers in time or linearly implicit solvers in time. For example, in Maple, the linearly implicit Rosenbrock stiff ODE solver is very good at handling these systems. The proposed approach is shown to be effective in solving systems of DAEs through 5 examples in the next section.

3. RESULTS AND DISCUSSION: THE SINGLE STEP APPROACH APPLIED

Example 1: Index-1 DAE (1Algebraic and 1 Differential variable)

Consider the system below where *y* is the differential variable and *z* is the algebraic variable:

$$\frac{dy(t)}{dt} = -y(t)^2 + z(t) \tag{3.1}$$

$$\cos\left(y(t)\right) - \sqrt{z(t)} = 0 \tag{3.2}$$

For the example here, we will set the differential variable IC at

$$y(0) = 0.25 \tag{3.3}$$

Standard DAE solvers would need the consistent IC for *z*,

$$z(0) = \cos(0.25)^2 \approx 0.938791 \tag{3.4}$$

Standard solver packages might fail when the exact algebraic IC is not given (if the solver did not include initialization routines). Application of the single-step proposed approach is illustrated below.

A switch function is created shown as

$$T_H = \frac{1}{2}\left(1 + \tanh\left(1000(t-1)\right)\right) \tag{3.5}$$

This function is applied to the right hand side of the differential equation so that

$$\frac{dy(t)}{dt} = \left(-y(t)^2 + z(t)\right)\left(\frac{1}{2} + \frac{1}{2}\tanh\left(1000(t-1)\right)\right) \tag{3.6}$$

And a perturbation will be applied to the algebraic equation such that

$$\varepsilon\left(\sin\left(y(t)\right)\left(\frac{dy(t)}{dt}\right)+\frac{1\sqrt{z(t)}}{2z(t)}\frac{dz(t)}{dt}\right)=\cos\left(y(t)\right)-\sqrt{z(t)} \tag{3.7}$$

The system of Eqs. (3.6) & (3.7) can now be solved with an explicit (or linearly implicit) ODE solver. The IC of the algebraic variable does not have to be known *a priori*, but rather the combination of the switch function and perturbation will allow the IC for the algebraic variable to reach its consistent value during the first second of the solution because T_H will be 0, holding the differential variable constant during this initialization. After one second, T_H will switch to the value one (see Figure 2), and the solution of the complete system will begin based on the initialized conditions. A reasonable initial guess must still be provided for the starting value of *z*. The time of initialization must be removed from the solution time in order to achieve the real simulation time of the system. In the example cases, the initialization time is shown as negative time in the figures and the real simulation time of the problem is shown as positive.

Smaller values of ε will increase the accuracy of the converged value ensuring that the converged value can be used as a consistent IC, but a smaller ε will also increase the stiffness of the system. Applying larger ε will reduce the stiffness, but will also decrease the accuracy of the converged value. When ε becomes large enough, the converged value will no longer work as a consistent IC for the system of DAEs. The single step approach allows for a correction to the converged value at the end of initialization because of the continuous nature of the switch function. As the ODE variables begin to be solved, the converged value of the algebraic variables (as well as the IC for the differential variables) can correct to consistent values. This correction will introduce a small amount of error, but allows for a much more robust approach to solving by increasing the allowable converged value for the algebraic variables which will still solve the system.

For Example 1, the proposed approach with the switch function will solve the system with less than 0.001 error for all points after 0.1s of simulation time for converged values between $0.9377018 < z(0) < 0.9398806$. Even though the range of allowable converged values is small it greatly increases the robustness of the solver and allows for a much wider range of ε. These ranges are

dependent on the discreteness of the switch function used (q=1000 for the range above). Also, these ranges are only the allowable range of converged solutions and the allowable range of initial guesses will be much larger (see Figure 4).

For Example 1, if a starting guess of $z(0)$=0.8 was applied, the normal perturbation solution would require no larger than $\varepsilon = 1.1\times10^{-6}$ in order to obtain a solution for initialization that is consistent for the system of DAEs. However, the single-step approach can solve the same system with $\varepsilon = 0.1$ (or larger). Figure 3 shows a plot for the original two-step perturbation and simulation approach and the single-step solution. In the example discussed here, q is taken as 1000. The value of q will affect the discreteness of the switch function and can have an effect on the increased robustness. The proposed approach can be solved for many different initial guesses of the algebraic variables. Figure 4 shows the initialization period for 8 different initial algebraic guesses where $\varepsilon = 0.1$ and $t_j = 1$ for all cases. All of the guesses converge to the consistent IC of $z(0)$=0.938 and accurately solve the system of DAEs.

The perturbation value does affect the accuracy and convergence of the initialization. Smaller values of ε will allow the initialization to converge over shorter simulation time and when holding t_j constant, larger values of ε may obtain an initialization that is not accurate enough to satisfy the consistency condition. Figure 5 shows the initialization of Example 1 for several different perturbation values (ε=0.1, 0.05, 0.01, 0.001).

Example 2: Wu and White Problem

An example of a two equation system representing a thin film nickel hydroxide electrode described in Wu and White (2001) is studied during the charging process. This system can cause difficultly in determining consistent ICs. The system is represented by the equations (Methekar et al., 2011)

$$\frac{\rho V}{W}\frac{dy(t)}{dt} = \frac{j_1}{F} \tag{3.8}$$

$$j_1 + j_2 - i_{app} = 0 \tag{3.9}$$

Where,

$$j_1 = i_{o,1}\left[2(1-y(t))\exp\left(\frac{(z(t)-\phi_1)F}{2RT}\right) - 2y(t)\exp\left(-\frac{(z(t)-\phi_1)F}{2RT}\right)\right] \tag{3.10}$$

$$j_2 = i_{o,2}\left[\exp\left(\frac{(z(t)-\phi_2)F}{RT}\right) - \exp\left(-\frac{(z(t)-\phi_2)F}{RT}\right)\right] \tag{3.11}$$

The parameters of the system are given in Table 1. The differential variable *y* represents the nickel hydroxide mole fraction and the algebraic variable *z* represents the potential difference at the solid liquid interface.

In a discharged state, the mole fraction of the nickel hydroxide is estimated to be

$$y(0) = 0.05 \tag{3.12}$$

Under the algebraic constraint, the consistent IC for the potential must be

$$z(0) \approx 0.350236 \tag{3.13}$$

(Several imaginary roots are solutions to the algebraic constraint, but these are non-physical solutions, so they are not useful for the electrode equations shown here). When deviating from the consistent ICs, many initialization routines and solvers fail to obtain a solution. Table 2 shows the range of possible ICs for the algebraic variable that provide a solution for different solvers and approaches including the proposed single-step approach. Under normal conditions (no initialization), these solvers require initial guesses that are very close to the consistent values. The proposed single-step approach greatly widens the possible range of ICs. The solution for a full charge of the electrode is shown in Figure 6 (with an initial guess z(0)=0.7). For systems where the ICs are not easily available or obvious from the physical system, the expanded range of possible initial guesses is important for obtaining a solution. As mentioned before a compiled single procedure can be obtained for a nonlinear DAE system with this approach just like a procedure for ODEs. This helps improve the efficiency and robustness for inverse optimization problems (optimal control and parameter/state estimation).

Example 3: Implicit ODE converted to DAE solved with Explicit Solver

The proposed approach can also be used to solve implicit ODEs. Consider the implicit ODE:

$$\left(\frac{dy(t)}{dt}\right)^2+\left(\frac{dy(t)}{dt}\right)\left(y(t)+1\right)+y(t)=\cos\left(\frac{dy(t)}{dt}\right) \tag{3.14}$$

This problem cannot be directly solved using explicit solvers. When attempted, Maple states that IC for *dy/dt* is not known or the system cannot be converted to explicit ODE form. However, including a substitution

$$\frac{dy(t)}{dt}=z(t) \tag{3.15}$$

converts Eq. (3.14) into:

$$z(t)^2+z(t)\left(y(t)+1\right)+y(t)=\cos\left(z(t)\right) \tag{3.16}.$$

This adds a variable *z*, allowing the proposed approach to be used. The proposed approach is applied to Eq. (3.15) and (3.16) leading to:

$$\varepsilon\left(2z(t)\frac{dz(t)}{dt}+\frac{dz(t)}{dt}\left(y(t)+1\right)+z(t)\frac{dy(t)}{dt}+\frac{dy(t)}{dt}+\sin\left(z(t)\right)\frac{dz(t)}{dt}\right)=$$
$$z(t)^2+z(t)\left(y(t)+1\right)+y(t)-\cos\left(z(t)\right) \tag{3.17}$$

$$\frac{dy(t)}{dt}=z(t)\left(\frac{1}{2}+\frac{1}{2}\tanh\left(1000(t-1)\right)\right) \tag{3.18}$$

where q=1000 and t_j=1 for the switch function. The results for both initialization and simulation are shown in Figure 7 for the system solved using $\varepsilon=0.1$ with ICs of:

$$y(0)=0\ \ z(0)=0 \tag{3.19}$$

The converged value for *z*(0) from the initialization portion of the solver is 0.550.

Example 4: Partial Differential Equation Discretized to DAEs

When PDEs are discretized they can create a system of DAEs. Many of these systems consist of variables with explicit time derivatives (like concentration) and static variables (like potential which may not consist of time derivatives, but still changes with time because other variables in the system change with time). Consider the following set of PDEs:

$$\frac{\partial y}{\partial t} = \frac{\partial^2 y}{\partial x^2} - y(1+z) \tag{3.20}$$

$$\frac{\partial^2 z}{\partial x^2} = (1-y^2)\exp(-z) \tag{3.21}$$

With the boundary conditions:

$$\left.\frac{\partial y}{\partial x}\right|_{x=0} = 0 \quad y|_{x=1} = 1 \tag{3.22}$$

$$\left.\frac{\partial z}{\partial x}\right|_{x=0} = 0 \quad z|_{x=1} = 0 \tag{3.23}$$

Though $\partial z/\partial t$ is not present in the system, z changes with time as y changes with time. The model can be solved using numerical method of lines (White, 2010) which will discretize the spatial derivatives over a series of node points between the system's boundaries. When discretizing for the spatial variable the system becomes:

$$\frac{dy_i}{dt} = \frac{1}{h^2}(y_{i+1} - 2y_i + y_{i-1}) - y_i(1+z_i) \tag{3.24}$$

$$\frac{1}{h^2}(z_{i+1} - 2z_i + z_{i-1}) = (1-y_i^2)\exp(-z_i) \tag{3.25}$$

where h is the length between node points. And the boundary conditions become:

$$\frac{1}{2h}(3y_0 - 4y_1 + y_2) = 0 \quad y_{N+1} = 1 \tag{3.26}$$

$$\frac{1}{2h}\left(3z_0 - 4z_1 + z_2\right) = 0 \quad z_{N+1} = 0 \tag{3.27}$$

where N is the number of interior node points of the system. Converting to the proposed approach the system equations become:

$$\frac{dy_i}{dt} = \left[\frac{1}{h^2}\left(y_{i+1} - 2y_i + y_{i-1}\right) - y_i\left(1 + z_i\right)\right]\left(\frac{1}{2} + \frac{1}{2}\tanh\left(1000(t-1)\right)\right) \tag{3.28}$$

$$\begin{aligned} &\frac{-\varepsilon}{h^2}\left[\frac{dz_{i+1}}{dt} - 2\frac{dz_i}{dt} + \frac{dz_{i-1}}{dt} + 2y_i\frac{dy_i}{dt}\exp(-z_i) + \left(1 - y_i^2\right)\frac{dz_i}{dt}\exp(-z_i)\right] \\ &= \left(1 - y_i^2\right)\exp\left(-z_i\right) - \frac{1}{h^2}\left(z_{i+1} - 2z_i + z_{i-1}\right) \end{aligned} \tag{3.29}$$

And the boundary equations are:

$$\frac{-\varepsilon}{2h}\left[3\frac{dy_0}{dt} - 4\frac{dy_1}{dt} + \frac{dy_2}{dt}\right] = \frac{1}{2h}\left(3y_0 - 4y_1 + y_2\right) \quad -\varepsilon\frac{dy_{N+1}}{dt} = y_{N+1} - 1 \tag{3.30}$$

$$\frac{-\varepsilon}{2h}\left[3\frac{dz_0}{dt} - 4\frac{dz_1}{dt} + \frac{dz_2}{dt}\right] = \frac{1}{2h}\left(3z_0 - 4z_1 + z_2\right) \quad -\varepsilon\frac{dz_{N+1}}{dt} = z_{N+1} \tag{3.31}$$

For standard solvers the system cannot be solved for a large number N using explicit solvers (*i.e.*, one cannot convert this system of DAEs to explicit ODEs of the form $dy/dt = f$. Maple's dsolve, even with consistent ICs ($y_i(0)=1$ and $z_i(0)=0$), cannot solve the system for N greater than 5. Maple aims to convert the DAE system to an explicit ODE system of the form $dy/dt = f$ and fails for larger values of N. However using the single step proposed approach loosens the restrictions on the number of interior node points and increases the solving speed in Maple without having to use solvers involving Newton type iterations. Larger values of N may be required for higher accuracy and better convergence. Figure 8 shows the value of y at x=0, 1/3, 2/3, and 1 for N=2 and N=11 with ε=1x10^{-5}). The values at N=11 have converged to more accurate values, especially for values closer to x=0. Consistent ICs ($y_i(0)=1$ and $z_i(0)=0$) were used for Figure 8. The proposed approach can use the standard Maple dsolve approach to solve for more internal node points, without having to use direct DAE solvers that use Newton-Raphson type iterations. Even at

lower number of node points the proposed approach is faster than standard simulation techniques as it provides Maple with an approximate ODE that it can solve directly. At N=5 the proposed approach solves in 159ms using Maple dsolve, over an order of magnitude faster than the standard techniques inbuilt in Maple.

Example 5: Finite Difference Single Particle Lithium-ion Battery Model

Even when discretizing systems of PDEs that contain time derivatives in all the governing equations, the boundary conditions can yield algebraic equations. Our last example shows this case through a lithium-ion battery system. The single particle model (SPM) is used to describe the electrochemistry occurring in an intercalation based secondary battery (Santhanagopalan et al., 2006, Ramadesigan et al., 2012). The SPM has been used to model battery cycling and is a good model for batteries with thin electrodes and low charge and discharge rates (Pinson and Bazant, 2013, Northrop et al., 2014). The model tracks the diffusion of lithium inside the electrode particles of lithium-ion batteries governed by Fick's second law of diffusion:

$$\frac{\partial c_i}{\partial t} = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 D_i \frac{\partial c_i}{\partial r}\right) \quad i = p, n \tag{3.32}$$

With boundary conditions:

$$\left.\frac{\partial c}{\partial r}\right|_{r=0} = 0 \qquad \left.\frac{\partial c_i}{\partial r}\right|_{r=R_i} = -\frac{j_i}{D_i} \tag{3.33}$$

Where

$$j_i = \frac{\pm i_{app}}{a_i l_i F} = \pm 2k_i c_e^{0.5}(c_{i,\max} - c_{i,surf})^{0.5} c_{i,surf}^{0.5} \sinh\left(\frac{F(\Phi_i - U_i}{2RT}\right) \tag{3.34}$$

The parameters for the system are given in Table 3. The model can be solved using numerical method of lines which will discretize the spatial derivatives over a series of node points, N, within the particle, while the time derivative will remain. This discretization will create a DAE system with 2N differential

equations and 4 algebraic equations (the boundary conditions). The system can be solved for the concentration of lithium at every node point in the electrodes and then the electrode potentials, Φ, can be determined from the electrode surface concentrations. In a discharged state the initial concentration for lithium throughout the electrodes will be:

$$c_p(0) = 305.55 \; c_n(0) = 49503.11 \tag{3.35}$$

with units of mol/m^3.

This model system written in finite difference form (using second order central difference approach) and applying the single-step approach becomes

$$\frac{dc_{i,j}}{dt} = \frac{D_i}{j^2 h^2 R_i^2}\left[c_{i,j+1}\left(j^2 + j\right) + c_{i,j-1}\left(j^2 - j\right) - 2j^2 c_{i,j}\right]\left(\frac{1}{2} + \frac{1}{2}\tanh\left(1000(t-1)\right)\right) \tag{3.36}$$

With boundary conditions converted using second order 3 point forward and backward differences as

$$\frac{-\varepsilon}{2h}\left(4\frac{dc_{i,1}}{dt} - \frac{dc_{i,2}}{dt} - 3\frac{dc_{i,0}}{dt}\right) = 4c_{i,1} - c_{i,2} - 3c_{i,0} \tag{3.37}$$

$$\frac{-\varepsilon}{2h}\left(4\frac{dc_{i,N}}{dt} - \frac{dc_{i,N-1}}{dt} - 3\frac{dc_{i,N+1}}{dt}\right) = 4c_{i,N} - c_{i,N-1} - 3c_{i,N+1} - \frac{j_i}{D_i} \tag{3.38}$$

Results for the concentration values (five internal node points) are shown in Figure 9. Using the Rosenbrock stiff solver in Maple, the system can be solved for a complete 1C rate charge with up to 58 internal node points when not applying the proposed approach before the solver fails due to memory constraints. Note that Rosenbrock methods for DAE require additional constraints to satisfy the order requirement (Schneider, 1991, Hairer and Wanner, 1996). By using the proposed single-step approach, the same system under the same memory constraints can be solved for over 2,500 internal node points. The proposed approach reduces the computational burden on the solver and allows for larger systems to be solved thereby facilitating more accurate results. Table 4 shows the solving speed for simulating the lithium concentration throughout the electrode particles for a range of node points. The ICs for both cases

were given as shown in Eq. (3.35). The switch function was applied with t_j=1 and q=1000 (same as previous examples) and the perturbation value was $\varepsilon = 1\times10^{-5}$. The system was solved with a Rosenbrock solver under both the standard FD scheme and the proposed approach. The computational savings from the proposed approach allow larger systems to be solved. For a SPM with 50 internal node points (resulting in a system of 4 AEs and 100 ODEs) the proposed approach reduced computational time by two orders of magnitude.

4. DISCUSSION

The single step initialization and simulation approach has been tested on other electrochemically based battery models. The proposed approach was used on the porous electrode pseudo two dimensional model outlined in Northrop et al.(Northrop et al., 2011). For one complete constant power charge, a collocation of 15,6,15 (resulting in 123 AEs and 92 ODEs) was solved using $\varepsilon = 1\times10^{-4}$. The porous electrode pseudo two dimensional model and single-step approach was also used to solve dynamic discharging of a battery occurring from electric vehicle driving (Lawder et al., 2014). The dynamic current increased system stiffness causing other solvers to fail, especially when solving over long times and multiple cycles, but the single-step approach was able to overcome the continuously changing input current values.

Note that the method described should not be necessarily viewed as better or more robust compared to DASKR/IDA/DASSL/GEAR or RADAU. However, the proposed approach enables explicit solvers to handle nonlinear DAEs. As of today, Maple does not have a nonlinear DAE solver (all of its DAE solvers convert DAEs to explicit ODEs before solving). Similar situations are seen in large scale system simulators (DYMOLA). The proposed approach provides an iteration free alternative for solving nonlinear DAEs without knowing consistent initial conditions for the algebraic variable.

Additionally, the proposed approach has been used to solve index-2 systems of DAEs in some cases. Index-2 systems of DAEs can be approached by differentiating the perturbed AEs twice in order to

provide a system in which the perturbed equation is a function of the algebraic variable. Future work will include assessing the strength of extending the proposed single-step approach to index-2 systems of DAEs generally.

The proposed approach was tested under different numerical schemes. Table 5 give a summary of different numerical methods built into Maple and MATLAB as well as a FORTRAN solver that were used to solve Example 1 under the proposed approach. RODAS, a Rosenbrock 4^{th} order L stable method, was used in the FORTRAN code and the driver file is provided in Appendix A. This version of the Rosenbrock method can handle M matrices more efficiently (there is no need to substitute the ODE derivatives in the lhs). Note that Table 5 provides solvers that perform Newton iterations for solving nonlinear ODEs as well (MEBDFI, LSODE, ode15s) for completion's sake.

5. CONCLUSIONS

A new method is proposed to enable explicit (and linear implicit) ODE solvers to solve nonlinear DAEs. This method makes use of a switch function for the ODEs and combines both the initialization and simulation for systems of DAEs. The single-step approach decreases stiffness for initialization when compared to the two step perturbation initialization and system solving. In addition, the single-step method reduced the strict requirement on required initial guesses for algebraic ICs when compared to standard solvers, allowing consistency to be met under more cases. The continuous nature of the switch function allows for correction of inconsistent converged ICs that occurs under large perturbation values. This extension of allowable consistent converged ICs helps to expand the robustness of the single-step method. This single-step approach can be applied when a system of DAEs face difficulty with initialization under standard solvers without having to call a separate solver for initialization purposes only. Examples show the approach being applied in mathematical and physically representative situations. The approach has been tested for large, physically representative systems and has decreased the computational time required to study these systems. Importantly, the proposed approach provides an

iteration free alternative to stiff solvers like DASKR and RADAU for solving DAEs when consistent initial conditions are unknown.

6. ACKNOWLEDGEMENTS

The authors thank the United States Department of Energy (DOE) for the financial support for this work though the Advanced Research Projects Agency – Energy (ARPA-E) award #DE-AR0000275.

CAPTIONS

Table 1: Parameters for Nickel Hydroxide Electrode.

Table 2: Comparison of working ranges of initial algebraic guess for Example 2 using different solvers. The Maple dsolve approach uses Maple's rkf45 method. (Lawerence Berkeley National Lab, 2014, Maplesoft, 2015, Mathworks, 2015, Mathworks, 2015).

Table 3: Parameters for SPM.

Table 4: Computational time for solving concentration profiles in Example 5 using Maple's dsolve. Standard finite difference fails to solve SPM beyond N=50.

Table 5: List of methods used to solve the proposed approach.

Figure 1: Diagram of the proposed single-step approach.

Figure 2: Comparison of different q values for the switch function.

Figure 3: Comparison of Example 1 solution for two-step perturbation approach and proposed single-step approach.

Figure 4: Initialization portion of the single-step simulation showing different initial guesses for the algebraic variable (z) from Example 1.

Figure 5: Initialization portion of the single-step simulation for different perturbation values for the algebraic variable (z) from Example 1.

Figure 6: Solution to Example 2.

Figure 7: Solution to Example 3.

Figure 8: Solution to Example 4 for N=2 and N=11 at four different x-values.

Figure 9: SPM concentration solutions for N=5.

REFERENCES

Berzins, M., Dew, P. M. and Furzeland, R. M. Developing Software for Time-Dependent Problems Using the Method of Lines and Differential-Algebraic Integrators. Appl Numer Math 1989; 5(5): 375-397.

Boovaragavan, V., Ramadesigan, V., Panchagnula, M. V. and Subramanian, V. R. Continuum Representation for Simulating Discrete Events of Battery Operation. J Electrochem Soc 2010; 157(1): A98-A104.

Brown, P. N., Hindmarsh, A. C. and Petzold, L. R. Using Krylov Methods in the Solution of Large-Scale Differential-Algebraic Systems. Siam J Sci Comput 1994; 15(6): 1467-1488.

Brown, P. N., Hindmarsh, A. C. and Petzold, L. R. Consistent initial condition calculation for differential-algebraic systems. Siam J Sci Comput 1998; 19(5): 1495-1512.

Cash, J. R. Modified extended backward differentiation formulae for the numerical solution of stiff initial value problems in ODEs and DAEs. J Comput Appl Math 2000; 125(1-2): 117-130.

Cash, J. R. Efficient numerical methods for the solution of stiff initial-value problems and differential algebraic equations. P Roy Soc a-Math Phy 2003; 459(2032): 797-815.

Cash, J. R. and Karp, A. H. A Variable Order Runge-Kutta Method for Initial Valu Problems with Rapidly Varying Right-Hand Sides. Acm T Math Software 1990; 16(3): 21.

Cellier, F. E. and Kofman, E. Continuous System Simulation, Springer US 2006.

Dassault Systems. (2015). "DYMOLA." Retrieved Jan. 2015, 2015, from http://www.3ds.com/products-services/catia/capabilities/modelica-systems-simulation-info/dymola.

de Swart, J. J. B., Lioen, W. M. and van der Veen, W. A. (1998). Specification of PSIDE. Amsterdam, NL, National Research Insititute for Mathematicas and Computer Science (CWI) (The Netherlands)**:** 15.

Dew, P. M. and Walsh, J. E. A Set of Library Routines for Solving Parabolic Equations in One Space Variable. Acm T Math Software 1981; 7(3): 295-314.

Dormand, J. R. and Prince, P. J. A family of embedded Runge-Kutta formulae. J Comput Appl Math 1980; 6(1): 7.

Enright, W. H., Jackson, K. R., Norsett, S. P. and Thomsen, P. G. Interpolants for Runge-Kutta Formulas. Acm T Math Software 1986; 12(3): 193-218.

Garcia, J. A. G. A singlar Perturbation Approach to Modeling Closed Kinematic Chains. Rice University 2000.

Gear, C. W. Simultaneous Numerical Solution of Differential-Algebraic Equations. Ieee T Circuits Syst 1971; Ct18(1): 89-&.

Gopal, V. and Biegler, L. T. A successive linear programming approach for initialization and reinitialization after discontinuities of differential-algebraic equations. Siam J Sci Comput 1998; 20(2): 447-467.

Hairer, E. and Wanner, G. Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems, Springer 1996.

Hairer, E. and Wanner, G. Stiff differential equations solved by Radau methods. J Comput Appl Math 1999; 111(1-2): 93-111.

Hindmarsh, A. C. LSODE and LSODI, two new initial value ordinary differential equation solvers. ACM SIGNUM Newletters 1980; 15: 2.

Lamour, R. and Mazzia, F. Computation of consistent initial values for properly stated index 3 DAEs. Bit 2009; 49(1): 161-175.

Lawder, M. T., Northrop, P. W. C. and Subramanian, V. R. Model-based SEI Layer Growth and Capacity Fade Analysis for EV and PHEV Batteries and Drive Cycles. J Electrochem Soc 2014; 161(14): A2099-A2108.

Lawerence Berkeley National Lab. (2014). "Sundials IDA." Retrieved Novemeber, 2014, 2014, from https://computation.llnl.gov/casc/sundials/description/description.html#descr_ida.

Leimkuhler, B., Petzold, L. R. and Gear, C. W. Approximation Methods for the Consistent Initialization of Differential-Algebraic Equations. Siam J Numer Anal 1991; 28(1): 205-226.
Li, P. F., Li, Y. Y. and Seem, J. E. Consistent initialization of system of differential-algebraic equations for dynamic simulation of centrifugal chillers. J Build Perform Simu 2012; 5(2): 115-139.
Li, S. T. and Petzold, L. Software and algorithms for sensitivity analysis of large-scale differential algebraic systems. J Comput Appl Math 2000; 125(1-2): 131-145.
Maplesoft. (2015). "Maple." Retrieved Jan. 2015, 2015, from http://www.maplesoft.com/products/maple/.
Maplesoft. (2015). "Maple dsolve." Retrieved June, 2015, 2015, from http://www.maplesoft.com/support/help/maple/view.aspx?path=dsolve.
Mathworks. (2015). "Matlab ODE15i." Retrieved June, 2015, 2015, from http://www.mathworks.com/help/matlab/ref/ode15i.html.
Mathworks. (2015). "Matlab ODE15s." Retrieved June, 2015, 2015, from http://www.mathworks.com/help/matlab/ref/ode15s.html.
Methekar, R. N., Ramadesigan, V., Pirkle, J. C. and Subramanian, V. R. A perturbation approach for consistent initialization of index-1 explicit differential-algebraic equations arising from battery model simulations. Computers & Chemical Engineering 2011; 35(11): 2227-2234.
Michelsen, M. L. Application of Semi-Implicit Runge-Kutta Methods for Integration of Ordinary and Partial-Differential Equations. Chem Eng J Bioch Eng 1977; 14(2): 107-112.
Northrop, P. W. C., Ramadesigan, V., De, S. and Subramanian, V. R. Coordinate Transformation, Orthogonal Collocation, Model Reformulation and Simulation of Electrochemical-Thermal Behavior of Lithium-Ion Battery Stacks. J Electrochem Soc 2011; 158(12): A1461-A1477.
Northrop, P. W. C., Suthar, B., Ramadesigan, V., Santhanagopalan, S., Braatz, R. D. and Subramanian, V. R. Efficient Simulation and reformulation of Lithium-Ion Battery Models for enabling electric transportation. J Electrochem Soc 2014; 161(8): 9.
Pantelides, C. C., Gritsis, D., Morison, K. R. and Sargent, R. W. H. The Mathematical-Modeling of Transient Systems Using Differential Algebraic Equations. Computers & Chemical Engineering 1988; 12(5): 449-454.
Petzold, L. Differential-Algebraic Equations Are Not Odes. Siam J Sci Stat Comp 1982; 3(3): 367-384.
Petzold, L. R. (1982). A Description of DASSL: A Differential/Algebraic System Solver, Sandia National Lab.
Pinson, M. B. and Bazant, M. Z. Theory of SEI Formation in Rechargeable Batteries: Capacity Fade, Accelerated Aging and Lifetime Prediction. J Electrochem Soc 2013; 160(2): A243-A250.
Praprost, K. L. and Loparo, K. A. A stability theory for constrained dynamic systems with applications to electric power systems. Ieee T Automat Contr 1996; 41(11): 1605-1617.
Ramadesigan, V., Northrop, P. W. C., De, S., Santhanagopalan, S., Braatz, R. D. and Subramanian, V. R. Modeling and Simulation of Lithium-Ion Batteries from a Systems Engineering Perspective. J Electrochem Soc 2012; 159(3): R31-R45.
Reissig, G., Boche, H. and Barton, P. I. On inconsistent initial conditions for linear time-invariant differential-algebraic equations. Ieee Transactions on Circuits and Systems I-Fundamental Theory and Applications 2002; 49(11): 1646-1648.
Santhanagopalan, S., Guo, Q. Z., Ramadass, P. and White, R. E. Review of models for predicting the cycling performance of lithium ion batteries. J Power Sources 2006; 156(2): 620-628.
Schneider, C. Rosenbrock-Type Methods Adapted to Differential-Algebraic Systems. Math Comput 1991; 56(193): 201-213.
Schwalbe, D., Kooijman, H. and Taylor, R. Solving stiff differential equations and differential algebraic systems with Maple V. Mapletech 1996; 3(2): 47-53.
Shampine, L. F. Numerical Solution of Ordinary Differential Equations. New York, Chapman & Hall 1994.
Shampine, L. F. Solving 0=F(t, y(t), y'(t)) in Matlab. Journal of Numerical Mathematics 2002; 10(4): 19.

Shampine, L. F. and Corless, R. M. Initial value problems for ODEs in problem solving environments. J Comput Appl Math 2000; 125(1-2): 31-40.

Shampine, L. F. and Reichelt, M. W. The MATLAB ODE suite. Siam J Sci Comput 1997; 18(1): 1-22.

Shampine, L. F., Reichelt, M. W. and Kierzenka, J. A. Solving index-I DAEs in MATLAB and Simulink. Siam Rev 1999; 41(3): 538-552.

Susuki, Y., Hikiliara, T. and Chiang, H. D. Discontinuous dynamics of electric power system with dc transmission: A study on DAE system. Ieee T Circuits-I 2008; 55(2): 697-707.

Taylor, R. Engineering Computing with Maple: Solution of PDEs via the Method of Lines. CACHE News 1999; 49: 5-8.

VanKeken, P. E., Yuen, D. A. and Petzold, L. R. DASPK: A new high order and adaptive time-integration technique with applications to mantle convection with strongly temperature- and pressure-dependent rheology. Geophys Astro Fluid 1995; 80(1-2): 57-74.

Verner, J. H. Explicit Runge-Kutta Methods with Estimates of Local Truncation Error. Siam J Numer Anal 1978; 15(4): 772-790.

White, R. E. S., Venkat R. Computational Methods in Chemical Engineering with Maple. Berlin, Springer-Verlag 2010.

Wolfram. (2014). "NDsolve." Retrieved November, 2014, 2014, from http://reference.wolfram.com/language/ref/NDSolve.html.

Wu, B. and White, R. E. An initialization subroutine for DAEs solvers: DAEIS. Computers & Chemical Engineering 2001; 25(2-3): 301-311.

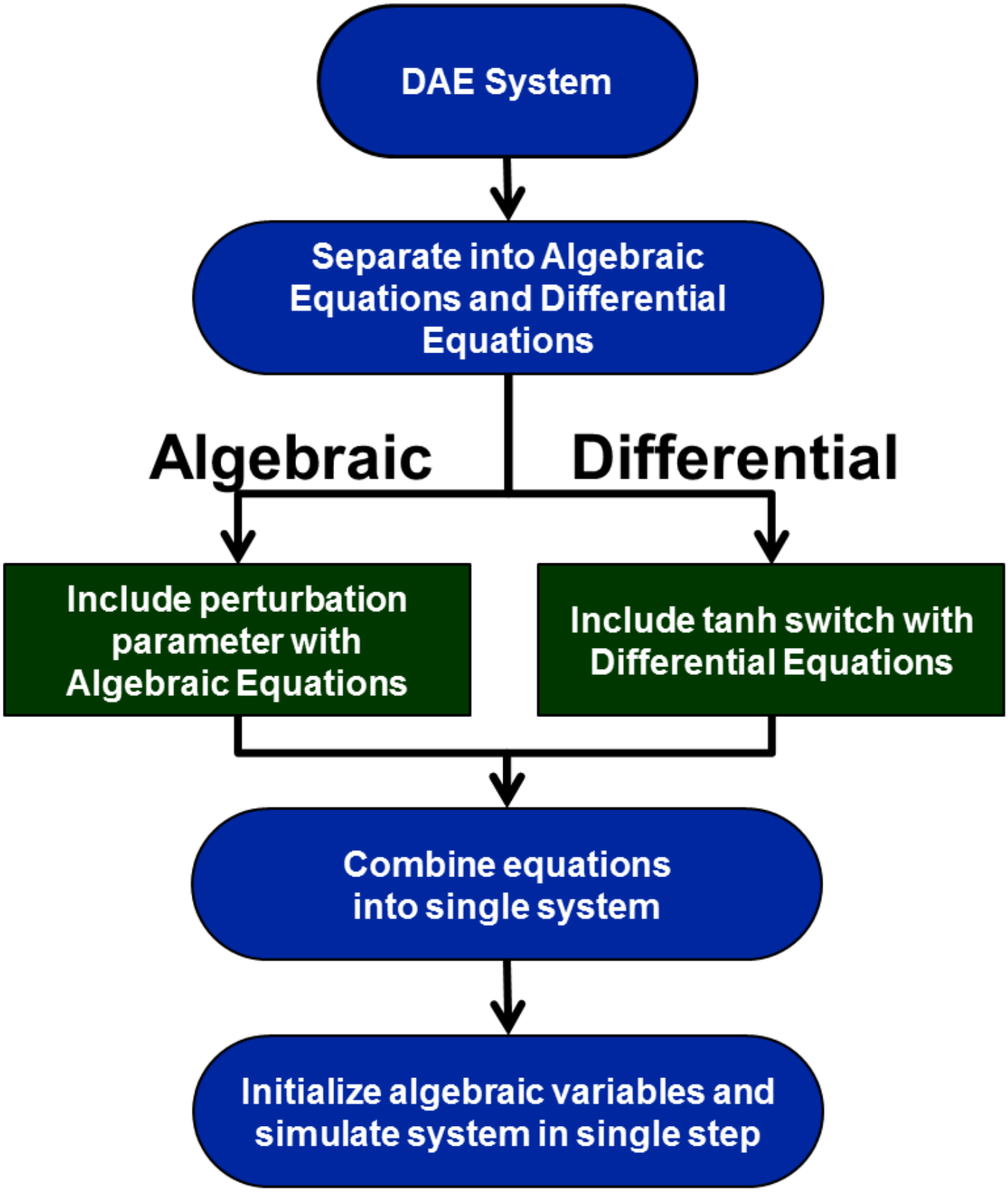

DAE System
Separate into Algebraic Equations and Differential Equations
Algebraic
Differential
Include perturbation parameter with Algebraic Equations
Include tanh switch with Differential Equations
Combine equations into single system
Initialize algebraic variables and simulate system in single step

Fig1

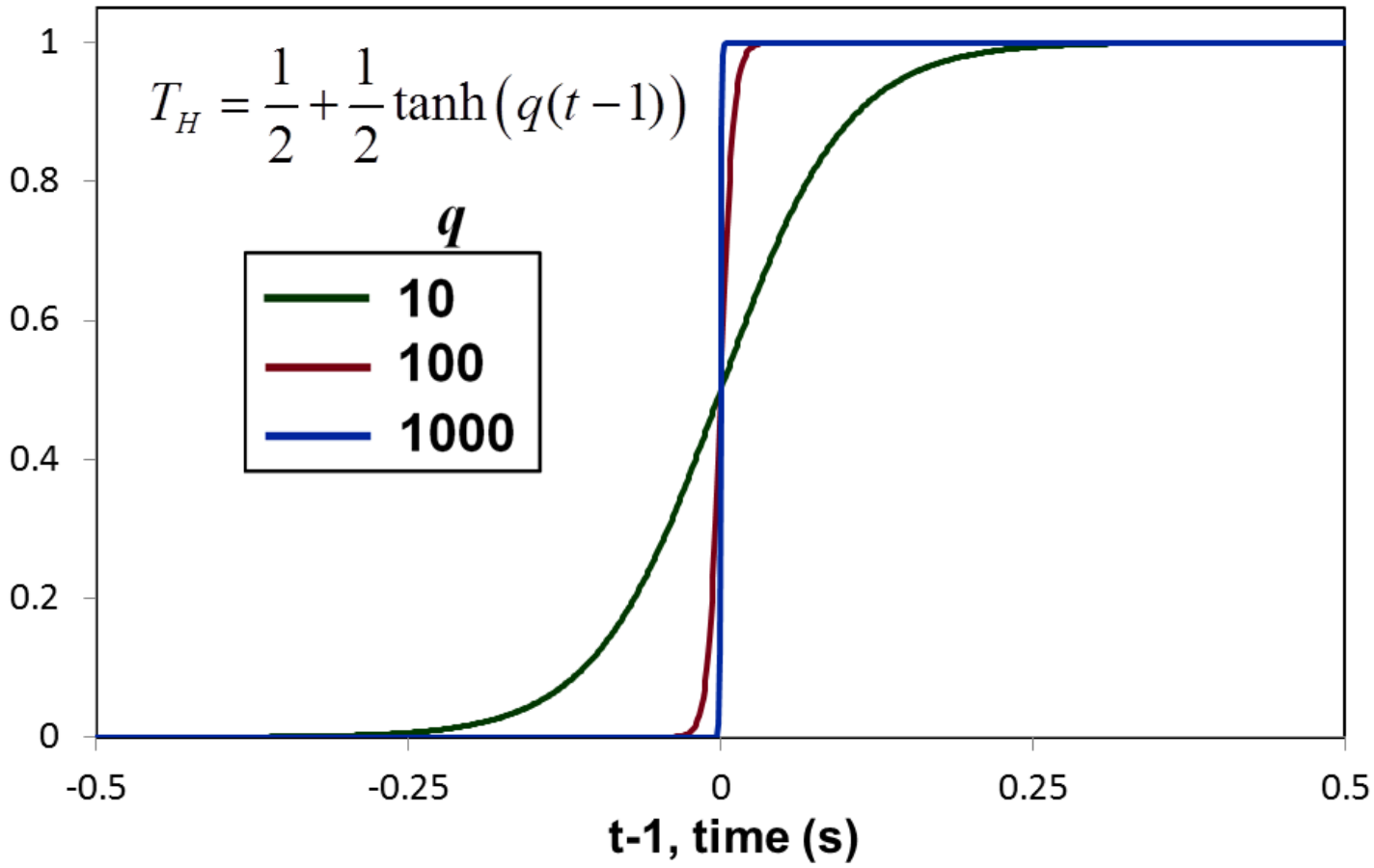

$T_H = \frac{1}{2} + \frac{1}{2}\tanh\left(q(t-1)\right)$
q
10
100
1000
1
0.8
0.6
0.4
0.2
0
-0.5
-0.25
0
0.25
0.5
t-1, time (s)

Fig2

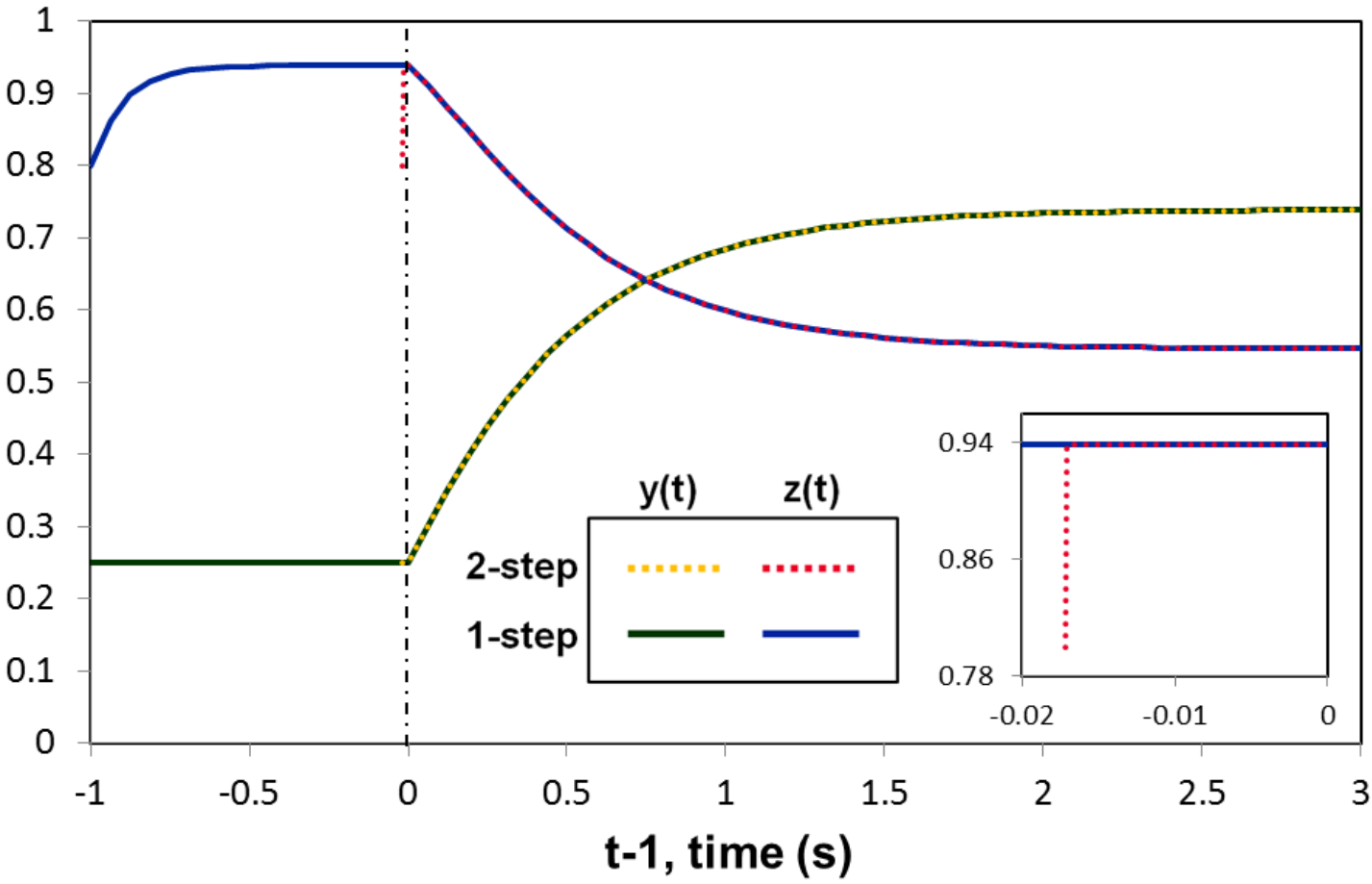

1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-1
-0.5
0
0.5
1
1.5
2
2.5
3
t-1, time (s)
y(t)
z(t)
2-step
1-step
0.94
0.86
0.78
-0.02
-0.01
0

Fig3

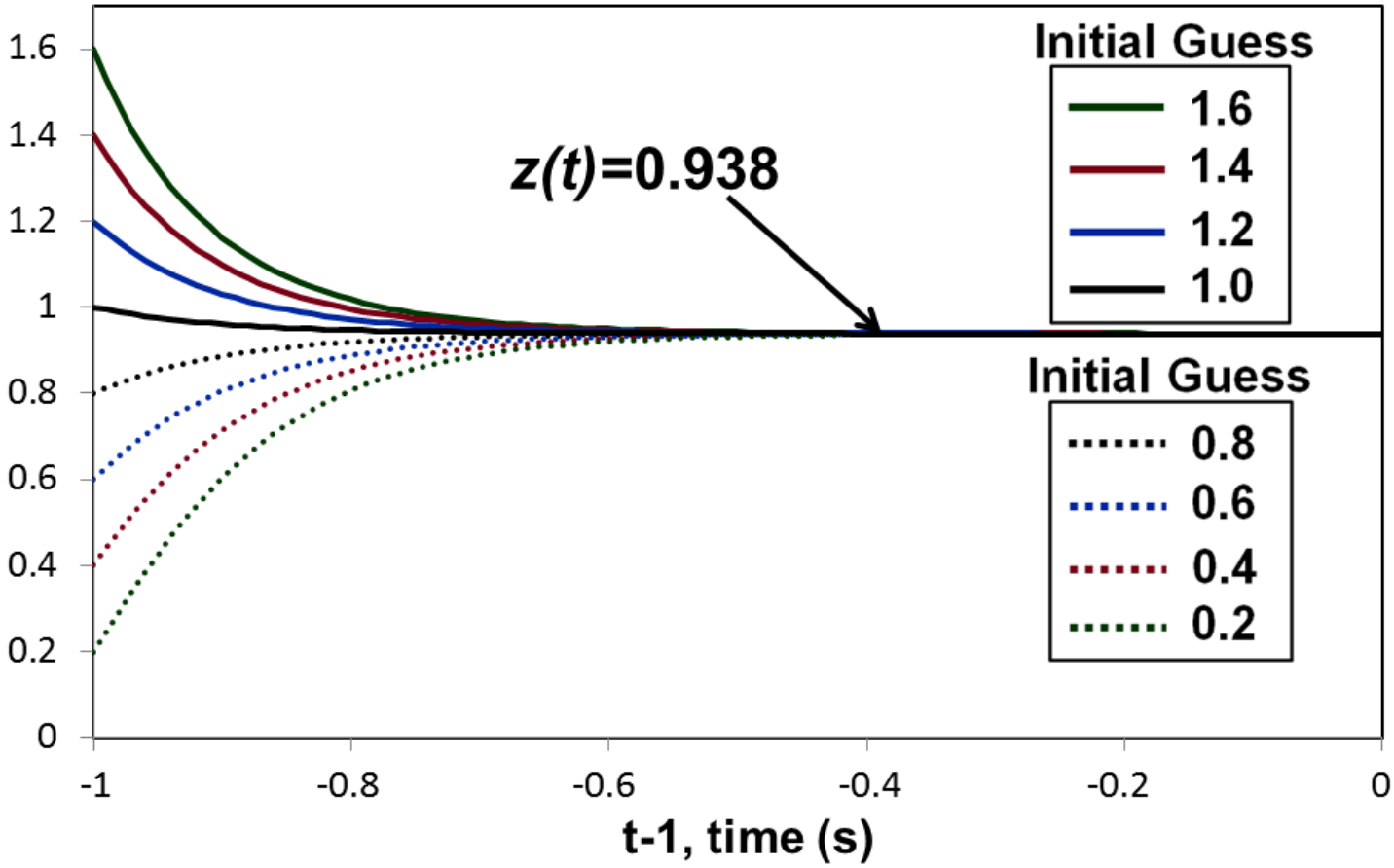

1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-1
-0.8
-0.6
-0.4
-0.2
0
t-1, time (s)
z(t)=0.938
Initial Guess
1.6
1.4
1.2
1.0
Initial Guess
0.8
0.6
0.4
0.2

Fig4

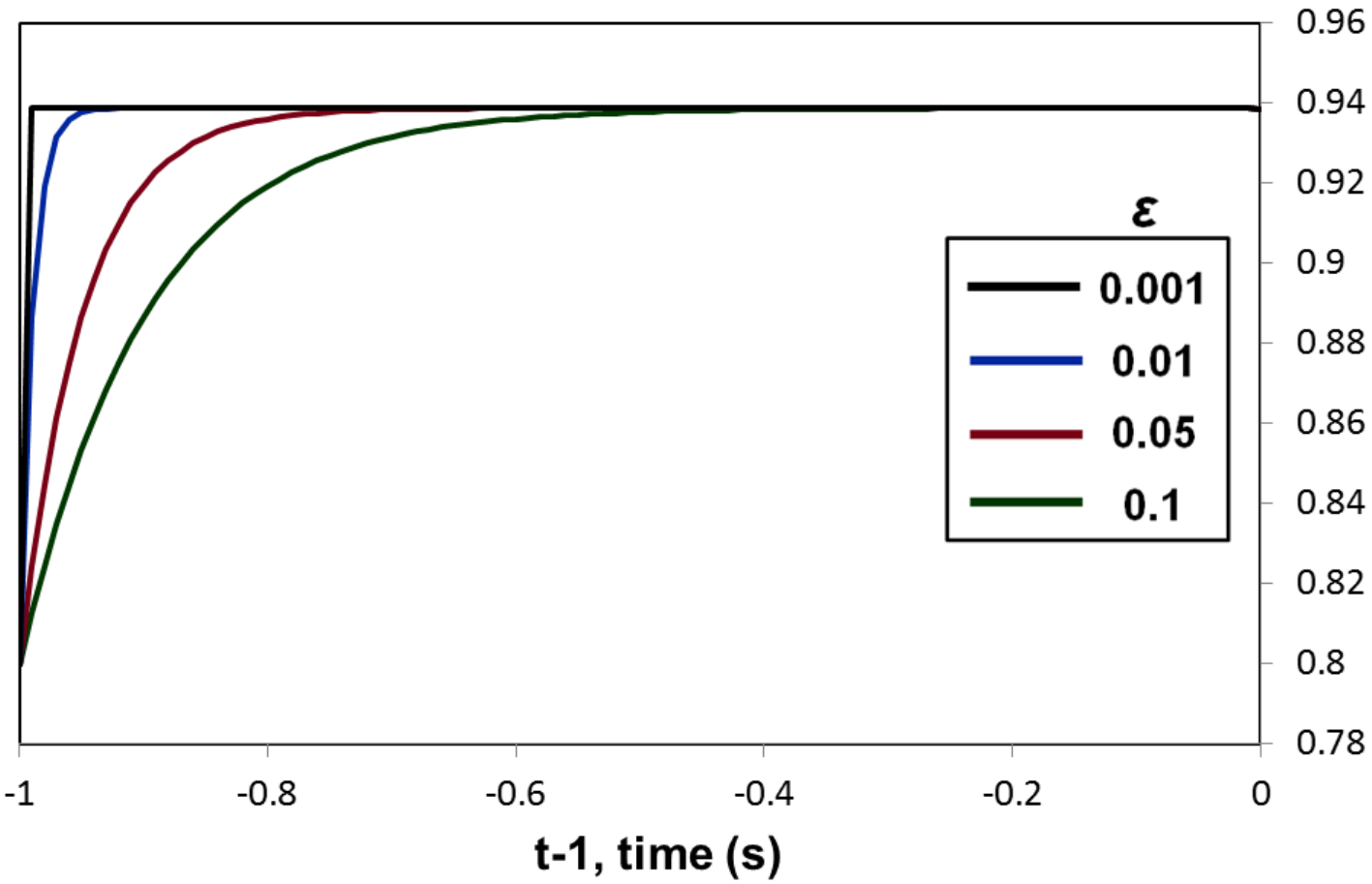

0.96
0.94
0.92
0.9
0.88
0.86
0.84
0.82
0.8
0.78
ε
0.001
0.01
0.05
0.1
-1
-0.8
-0.6
-0.4
-0.2
0
t-1, time (s)

Fig5

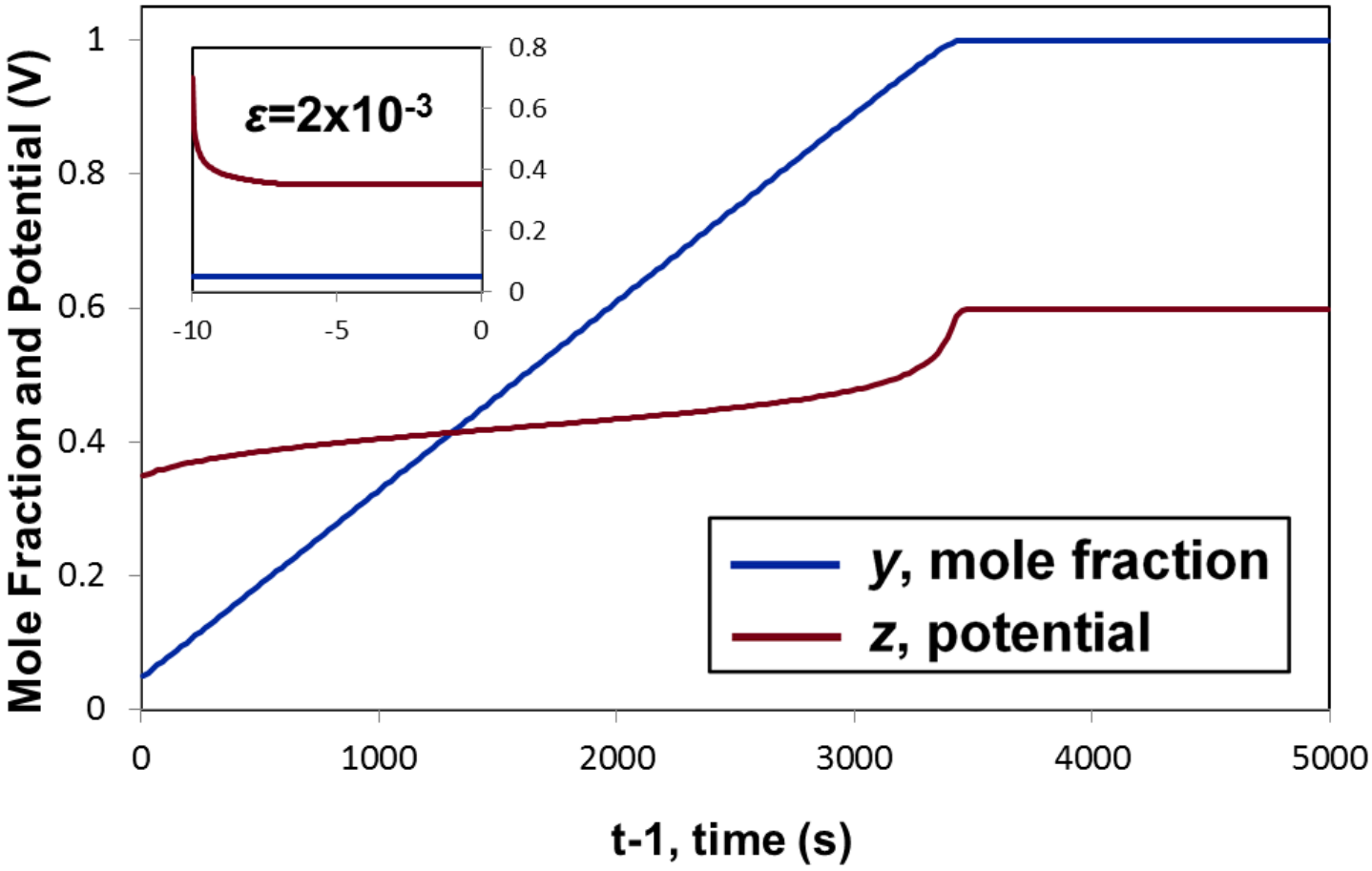

Mole Fraction and Potential (V)
t-1, time (s)
ε=2x10-3
y, mole fraction
z, potential
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
-10
-5
0
0.8
0.6
0.4
0.2
0

Fig6

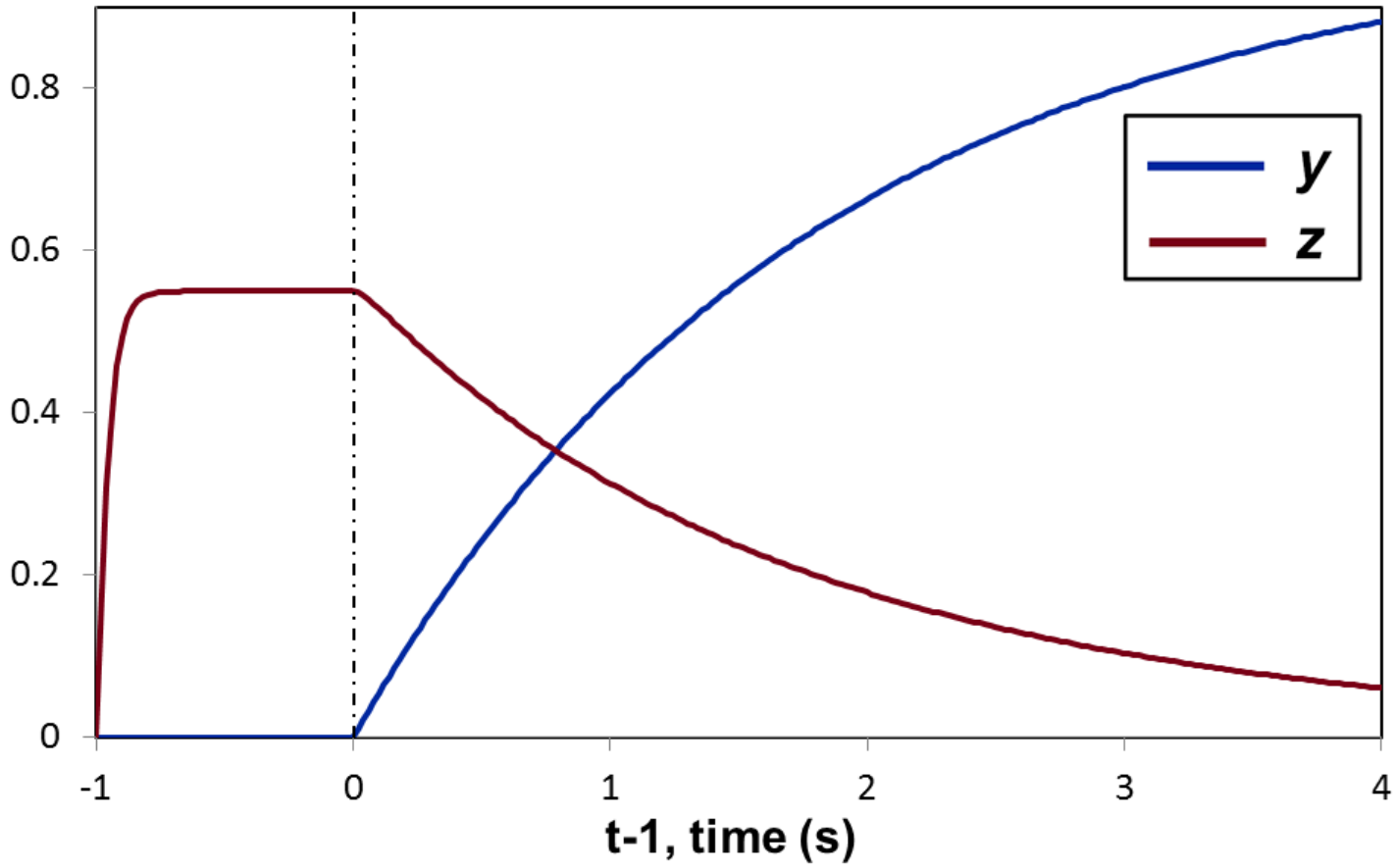


Fig7

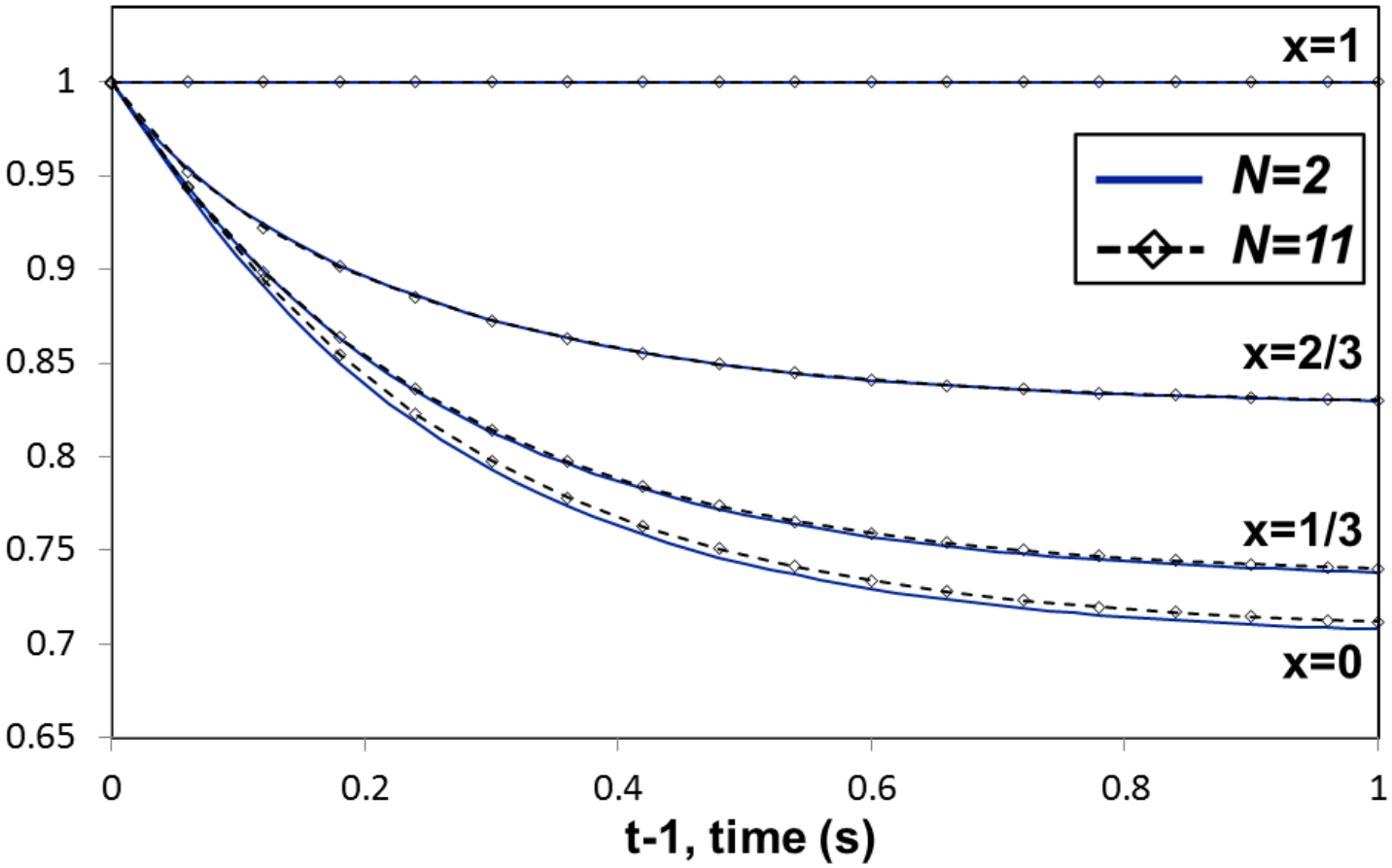

x=1
N=2
N=11
x=2/3
x=1/3
x=0
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0
0.2
0.4
0.6
0.8
1
t-1, time (s)

Fig8

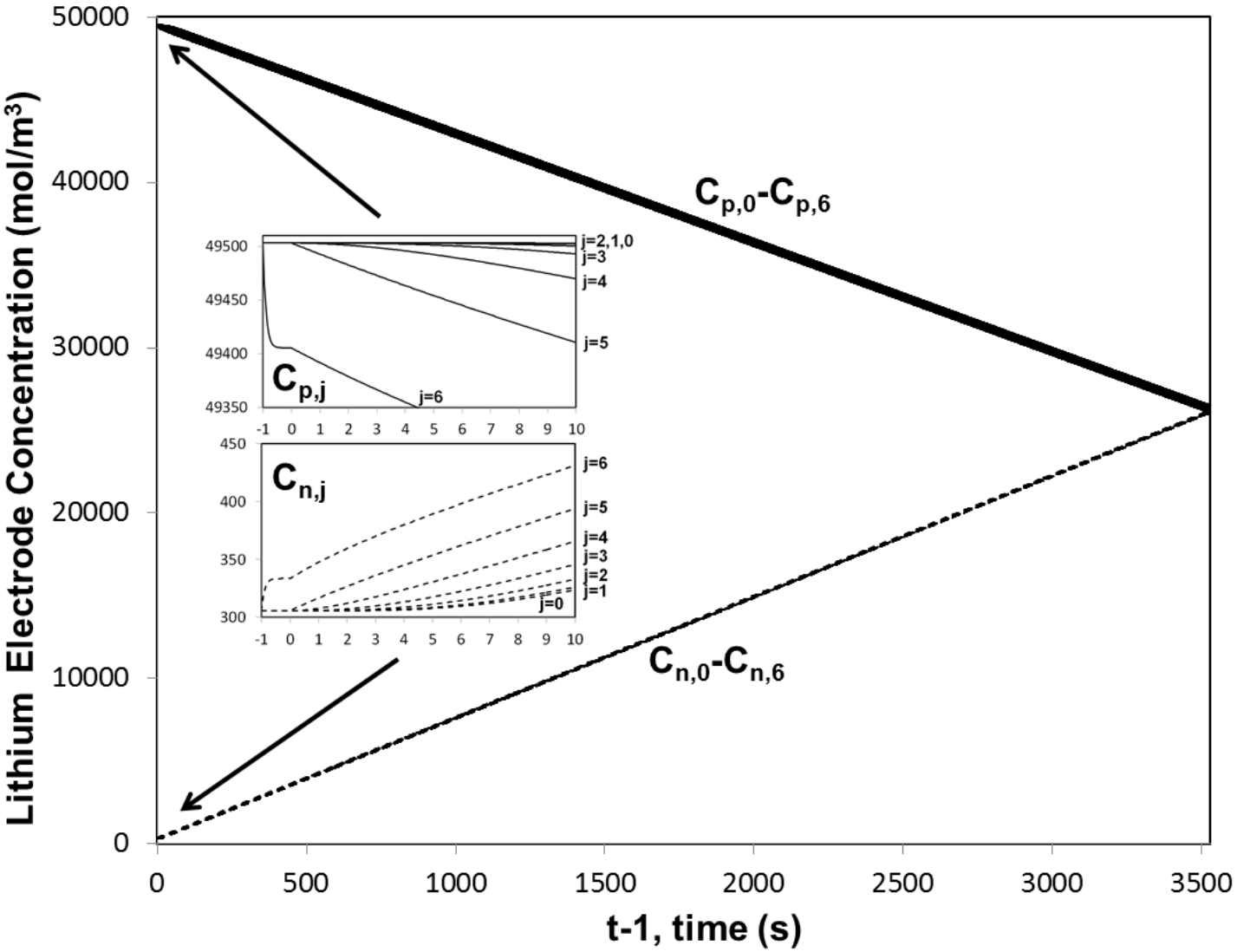

Lithium Electrode Concentration (mol/m3)
t-1, time (s)
Cp,0-Cp,6
Cn,0-Cn,6
Cp,j
Cn,j
j=2,1,0
j=3
j=4
j=5
j=6
j=0
j=1
j=2

Fig9

Parameters			
Symbol	**Parameter**	**Value**	**Units**
F	Faraday Constant	96487	C/mol
R	Gas Constant	8.3143	J/(mol K)
T	Temperature	303.15	K
ϕ_1	Equilibrium potential	0.420	V
ϕ_2	Equilibrium potential	0.303	V
W	Mass of active material	92.7	g
V	Volume	$1x10^{-5}$	m^3
i_{o1}	Exchange current density	$1x10^{-4}$	A/cm^2
i_{o2}		$1x10^{-10}$	A/cm^2
i_{app}	Applied current	$1x10^{-5}$	A/cm^2
ρ	Density	3.4	g/cm^3

Table 1

Solver	Algebraic Range
Maple dsolve	z(0)=0.3502359, Exact conditions required
MATLAB ode15i	0.342 < z(0) < 0.365
MATLAB ode15s	0.271 < z(0) < 0.474
SUNDIALS IDA	-1.27 < z(0) < 1.87
Proposed Approach Maple dsolve	-9.13 < z(0) < 9.85

Table 2

Parameters				
Symbol	**Parameter**	**Value**		**Units**
F	Faraday Constant	96487		C/mol
R	Gas Constant	8.3143		J/(mol K)
T	Temperature	303.15		K
c_e	Electrolyte concentration	1000		mol/m^3
i_{app}	Applied current	1		C
		Cathode (p)	**Anode (n)**	
D	Solid phase Diffusion Coefficient	1.0×10^{-14}	3.9×10^{-14}	m^2/s
a	particle surface area to volume	8.85×10^{5}	7.236×10^{5}	m^2/m^3
c^{max}	Maximum lithium concentration	51555	30555	mol/m^3
l	Cell thickness	80×10^{-6}	88×10^{-6}	m
R	Electrode particle radius	2×10^{-6}	2×10^{-6}	m
k_o	Reaction rate	2.334×10^{-11}	5.0307×10^{-11}	$m^{2.5}/(mol^{0.5} s)$
U	Overpotential	is a function of state-of-charge (Northrop et al., 2011)		

Table 3

Internal Node points	Standard FD Time (ms)	Proposed Single-Step Time (ms)
5	111	55
25	3340	94
50	23715	200
100	N/A	404
500	N/A	4377

Table 4

Method	Stiff solver	Computational Time (ms)	Reference
Maple mebdfi	Yes	169	(Cash, 2000)
Maple rkf45	No	128	(Enright et al., 1986, Shampine and Corless, 2000)
Maple ck45	No	128	(Enright et al., 1986, Cash and Karp, 1990)
Maple gear	Yes	125	(Gear, 1971)
Maple dverk78	No	125	(Verner, 1978, Dormand and Prince, 1980)
Maple lsode	Yes	125	(Hindmarsh, 1980)
Maple rosenbrock	Yes	131	(Hairer and Wanner, 1996)
MATLAB ode15i	No	35.8	(Shampine, 2002)
MATLAB ode15s	Yes	62.4	(Dormand and Prince, 1980, Shampine, 1994)
MATLAB ode23s	Yes	119	(Dormand and Prince, 1980, Shampine and Reichelt, 1997)
FORTRAN RODAS	Yes	154	(Hairer and Wanner, 1996)

Table 5

Appendix A

Code from Maple, MATLAB, and Fortran for Example 1 from "Extending Explicit and Linearly Implicit ODE Solvers for Index-1 DAEs."

1. Maple Code (using dsolve's rkf45)

Use y1, y2, etc. for all differential variables and z1, z2, etc. for all algebraic variables

```
> restart;
> with(plots):
```

Enter all ODEs in eqode

```
> eqode:=[diff(y1(t),t)=-y1(t)^2+z1(t)];
```

$$eqode := \left[\frac{d}{dt}\,y1(t) = -y1(t)^2 + z1(t)\right]$$

Enter all AEs in eqae

```
> eqae:=[cos(y1(t))-z1(t)^0.5=0];
```

$$eqae := [\cos(y1(t)) - z1(t)^{0.5} = 0]$$

Enter all initial conditions for differential variables in icodes

```
> icodes:=[y1(0)=0.25];
```

$$icodes := [y1(0) = 0.25]$$

Enter all intial conditions for algebraic variables in icaes

```
> icaes:=[z1(0)=0.8];
```

$$icaes := [z1(0) = 0.8]$$

Enter parameters for perturbation value (epsilon), switch function (q and tint), and runtime (tf)

```
> pars:=[epsilon=0.1,q=1000,tint=1,tf=5];
```

$$pars := [\mu = 0.1,\ q = 1000,\ tint = 1,\ tf = 5]$$

Choose solving method (1 for explicit, 0 for implicit)

```
> Xexplicit:=1:
```

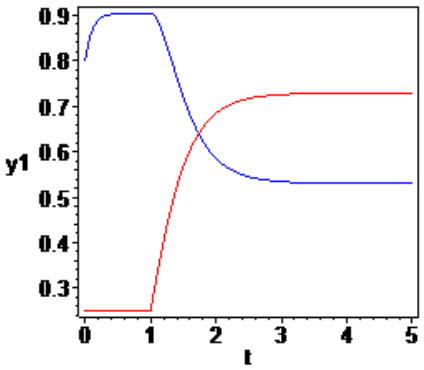
Standard solver requires IC z(0)=0.938791 or else it will fail

```
> solx:=dsolve({eqode[1],eqae[1],icodes[1],icaes[1]},numeric);
Error, (in dsolve/numeric/DAE/checkconstraints) the initial conditions do not satisfy the algebraic
constraints error = .745e-1, tolerance = .559e-6, constraint = cos(y1(t))-z1(t)^.50000000000000000000
> ff:=subs(pars,1/2+1/2*tanh(q*(t-tint)));
```

$$ff := \frac{1}{2} + \frac{1}{2}\tanh(1000\,t - 1000)$$

```
> NODE:=nops(eqode):NAE:=nops(eqae):
> for XX from 1 to NODE do
> EQODE||XX:=lhs(eqode[XX])=rhs(eqode[XX])*ff:
> end do:
> for XX from 1 to NAE do
> EQAE||XX:=subs(pars,-epsilon*(diff(rhs(eqae[XX])-lhs(eqae[XX]),t))=rhs(eqae
> [XX])-lhs(eqae[XX])):
> end do:
> Dvars1:={seq(diff(z||x(t),t)=D||x,x=1..NAE)}:
> Dvars2:={seq(rhs(Dvars1[x])=lhs(Dvars1[x]),x=1..NAE)}:
> icsn:=seq(subs(y||x(0)=y||x(t),icodes[x]),x=1..NODE),seq(subs(z||x(0)=
> z||x(t),icaes[x]),x=1..NAE):
> for j from 1 to NAE do
```

```
> EQAEX||j:=subs(Dvars1,eqode,icsn,Dvars2,lhs(EQAE||j))=rhs(EQAE||j):
> end do:
> Sys:={seq(EQODE||x,x=1..NODE),seq(EQAEX||x,x=1..NAE),seq(icodes[x],x=1
> ..NODE),seq(icaes[x],x=1..NAE)}:
> if Xexplicit=1 then
> sol:=dsolve(Sys,numeric):
> else
> sol:=dsolve(Sys,numeric,stiff=true,implicit=true):
> end if:
# Plotting Results
> for XX from 1 to NODE do
> a||XX:=odeplot(sol,[t,y||XX(t)],0..subs(pars,tf),color=red):
> end do:
> for XX from NODE+1 to NODE+NAE do
> a||XX:=odeplot(sol,[t,z||(XX-NODE)(t)],0..subs(pars,tf),color=blue):
> end do:
> display(seq(a||x,x=1..NODE+NAE),axes=boxed);
```



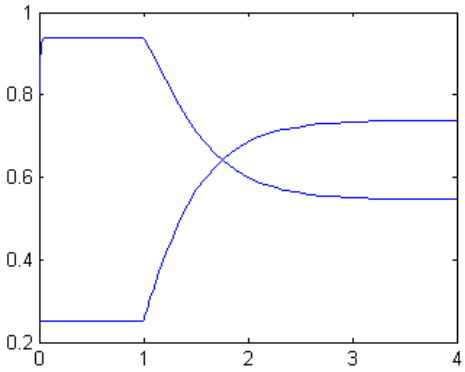
End Maple Code

2. MATLAB code (using ode15s)

Example 1 has been converted into a useable form for ode15s

```
function CCS3s
clear
clf
clc
hold off
tsp = 4;
tspan=[0 tsp];
%Extra dummy variable y(2)=dy/dt has been added so that y(1)=y; y(2)=dy/dt;
%y(3)=z
y0 = [0.25,0,0.8];
Nels = 3;
M = [1 0 0;0 0 0;0 0 1;];
options=odeset('Mass',M);
[T,Y]=ode15s(@MEQS,tspan,y0,options);
hold on
for i=1:2:Nels
plot(T,Y(:,i));
end
return
function [res]=MEQS(t,y)
```

```
tj=1;
q=1000;
epsilon=0.01;
ff=(1/2+1/2*tanh(q*(t-tj)));
%Converting the variables from Example 1, y->y(1), z->y(3), and dy/dt->y(2)
%y(2) must be used because ode15s must be of the form Mf(t,y')=f(t,y)
%Based on the Mass Function, M, Eq1 will equal the derivative of y(1), Eq2
%will equal zero, and Eq3 will equal the derivative of y(3)
Eq1=y(2);
Eq2=y(2)-(-y(1)^2+y(3))*ff;
Eq3=-2*y(2)*(y(3)^0.5)*sin(y(1))+2*cos(y(1))*(y(3)^0.5)/epsilon-2*y(3)/epsilon;
res = [Eq1;Eq2;Eq3;];
return
```



End Matlab Code

3. Fortran Code (using RODAS solver which is a Rosenbrock method solver)

In order to run the Fortran driver, you will also need download the RODAS solver, DECSOL linear algebra routines, and DC_DECSOL subroutines which are available free at http://www.unige.ch/~hairer/software.html

This code was compile using Compaq Visual Fortran 6

```
C * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C --- DRIVER FOR ROSENBROCK CODE RODAS
C * * * * * * * * * * * * * * * * * * * * * * * * * * * *
c link dr_rodas rodas decsol dc_decsol
c link dr_rodas rodas lapack lapackc dc_lapack
        IMPLICIT REAL*8 (A-H,O-Z)
C --- PARAMETERS FOR RODAS (FULL JACOBIAN)
        PARAMETER (ND=2,LWORK=6*ND*ND+14*ND+20,LIWORK=3*ND+20)
C --- DECLARATIONS
        DIMENSION Y(ND),WORK(LWORK),IWORK(LIWORK)
        EXTERNAL FEQN,JAC,SOLOUT,MAS,FIC
C --- DIMENSION OF THE SYSTEM
        N=2
C --- PROBLEM IS AUTONOMOUS
        IFCN=0
C --- COMPUTE THE JACOBIAN ANALYTICALLY
        IJAC=1
C --- JACOBIAN IS A FULL MATRIX
        MLJAC=N
C --- DIFFERENTIAL EQUATION IS IN EXPLICIT FORM
        IMAS=1
        MLMAS=N
C --- OUTPUT ROUTINE IS USED DURING INTEGRATION
```

```
        IOUT=1
C --- INITIAL VALUES
        X=0.0D0
        CALL FIC(N,Y)
C --- ENDPOINT OF INTEGRATION
        XEND=5.0D0
C --- REQUIRED TOLERANCE
        RTOL=1.0D-6
        ATOL=1.0D-6
        ITOL=0
C --- INITIAL STEP SIZE
        H=1.0D-6
C --- SET DEFAULT VALUES
        DO 10 I=1,20
        IWORK(I)=0
  10    WORK(I)=0.D0
C --- CALL OF THE SUBROUTINE RODAS
        CALL RODAS(N,FEQN,IFCN,X,Y,XEND,H,
     &                  RTOL,ATOL,ITOL,
     &                  JAC,IJAC,MLJAC,MUJAC,FVPOL,IDFX,
     &                  MAS,IMAS,MLMAS,MUMAS,
     &                  SOLOUT,IOUT,
     &                  WORK,LWORK,IWORK,LIWORK,RPAR,IPAR,IDID)
C --- PRINT FINAL SOLUTION
        WRITE (6,99) X,Y(1),Y(2)
 99     FORMAT(1X,'X =',F5.2,'    Y =',2E18.10)
C --- PRINT STATISTICS
        WRITE (6,90) RTOL
 90     FORMAT('       rtol=',D8.2)
        WRITE (6,91) (IWORK(J),J=14,20)
 91     FORMAT(' fcn=',I5,' jac=',I4,' step=',I4,
     &        ' accpt=',I4,' rejct=',I3,' dec=',I4,
     &        ' sol=',I5)
        STOP
        END
C
        SUBROUTINE SOLOUT (NR,XOLD,X,Y,CONT,LRC,N,RPAR,IPAR,IRTRN)
C --- PRINTS SOLUTION
        IMPLICIT REAL*8 (A-H,O-Z)
        DIMENSION Y(N),CONT(LRC)
        COMMON /INTERN/XOUT
        IF (NR.EQ.1) THEN
           WRITE (6,99) X,Y(1),Y(2),NR-1
           XOUT=0.2D0
        ELSE
           IF (X.GE.XOUT) THEN
              Y1=CONTRO(1,XOUT,CONT,LRC)
              Y2=CONTRO(2,XOUT,CONT,LRC)
              WRITE (6,99) XOUT,Y1,Y2,NR-1
              XOUT=XOUT+0.2D0
           END IF
        END IF
 99     FORMAT(1X,'X =',F5.2,'    Y =',2E18.10,'    NSTEP =',I4)
        RETURN
        END
C

      SUBROUTINE FEQN(N,X,Y,F,RPAR,IPAR)
!=============================================================
      IMPLICIT REAL*8 (A-H,O-Z)

      DIMENSION Y(N),F(N)

      F(1) = (0.5D0+0.5D0*tanh(1000.D0*X-1000.D0))*(-1.D0*Y(1)**2+Y(2))
```

```
      F(2) = cos(Y(1))-1.D0*Y(2)**0.5D0

      RETURN
      END


      SUBROUTINE JAC(N,X,Y,DFY,LDFY,RPAR,IPAR)
!=======================================================================
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N),DFY(LDFY,N)

      DFY(1,1) = -2.D0*(0.5D0+0.5D0*tanh(1000.D0*X-1000.D0))*Y(1)
      DFY(1,2) = 0.5D0+0.5D0*tanh(1000.D0*X-1000.D0)
      DFY(2,1) = -sin(Y(1))
      DFY(2,2) = -0.5D0/Y(2)**0.5D0

      RETURN
      END


      SUBROUTINE MAS(N,AM,LMAS,RPAR,IPAR)
!=======================================================================
      IMPLICIT REAL*8 (A-H,O-Z)
      DOUBLE PRECISION AM(LMAS,N),Y(N)

      AM(1,1) = 1
      AM(1,2) = 0
      AM(2,1) = 0.247403959254523D-5
      AM(2,2) = 0.52704627669473D-5

      RETURN
      END


      SUBROUTINE FIC(N,Y)
!=======================================================================
      IMPLICIT REAL*8 (A-H,O-Z)
      DOUBLE PRECISION Y(N)

      Y(1) = 0.25D0
      Y(2) = 0.8D0

      RETURN
      END
```

```
X = 0.00    Y =  0.2500000000E+00  0.8000000000E+00    NSTEP =   0
X = 0.20    Y =  0.2500000000E+00  0.9387912809E+00    NSTEP =  25
X = 0.40    Y =  0.2500000000E+00  0.9387912809E+00    NSTEP =  26
X = 0.60    Y =  0.2500000000E+00  0.9387912809E+00    NSTEP =  27
X = 0.80    Y =  0.2500000000E+00  0.9387912809E+00    NSTEP =  28
X = 1.00    Y =  0.2503026871E+00  0.9386461326E+00    NSTEP =  55
X = 1.20    Y =  0.4061356666E+00  0.8439243822E+00    NSTEP =  87
X = 1.40    Y =  0.5215630910E+00  0.7517666333E+00    NSTEP =  91
X = 1.60    Y =  0.6007188478E+00  0.6805114595E+00    NSTEP =  94
X = 1.80    Y =  0.6525172790E+00  0.6313223198E+00    NSTEP =  96
X = 2.00    Y =  0.6854701462E+00  0.5992640819E+00    NSTEP =  98
X = 2.20    Y =  0.7060826415E+00  0.5789833505E+00    NSTEP = 100
X = 2.40    Y =  0.7188464484E+00  0.5663555956E+00    NSTEP = 102
X = 2.60    Y =  0.7267018158E+00  0.5585617516E+00    NSTEP = 104
X = 2.80    Y =  0.7315187328E+00  0.5537753161E+00    NSTEP = 105
X = 3.00    Y =  0.7344657681E+00  0.5508444084E+00    NSTEP = 106
X = 3.20    Y =  0.7362663926E+00  0.5490527710E+00    NSTEP = 107
X = 3.40    Y =  0.7373657096E+00  0.5479586296E+00    NSTEP = 108
X = 3.60    Y =  0.7380364367E+00  0.5472909451E+00    NSTEP = 109
X = 3.80    Y =  0.7384455256E+00  0.5468836683E+00    NSTEP = 110
X = 4.00    Y =  0.7386964050E+00  0.5466338846E+00    NSTEP = 111
X = 4.20    Y =  0.7388551462E+00  0.5464758318E+00    NSTEP = 112
X = 5.00    Y =  0.7390521643E+00  0.5462796619E+00
     rtol=0.10D-05
fcn=  712 jac= 112 step= 120 accpt= 112 rejct=  8 dec= 120 sol=  720
```

End Fortran Code

End of Appendix A