Author version

Published as:

Huang, Z., Sarojini, D., & Hwang, J. T. (2023). Efficient gradient-based optimization of differential-algebraic equation systems. In AIAA SCITECH 2023 Forum (Paper No. AIAA 2023-0532). https://doi.org/10.2514/6.2023-0532

BibTeX for this reference:

https://lsdolab.github.io/lsdo-publications/publication/huang2023efficient/

```
@inproceedings{huang2023efficient,
  author = {Zeyu Huang and Darshan Sarojini and John T. Hwang},
  title = {Efficient Gradient-Based Optimization of Differential-Algebraic Equation
    Systems},
  booktitle = {AIAA SCITECH 2023 Forum},
  year = {2023},
  doi = {10.2514/6.2023-0532},
  url = {https://doi.org/10.2514/6.2023-0532}
}
```

Efficient Gradient-Based Optimization of Differential-Algebraic Equation Systems

Conference Paper · January 2023

DOI: 10.2514/6.2023-0532

3 authors, including:

Darshan Sarojini
University of California, San Diego
45 PUBLICATIONS **206** CITATIONS

SEE PROFILE

John T. Hwang
University of California, San Diego
92 PUBLICATIONS **1,976** CITATIONS

SEE PROFILE

Efficient Gradient-Based Optimization of Differential-Algebraic Equation Systems

Zeyu Huang*, Darshan Sarojini†, and John Hwang ‡
University of California - San Diego, San Diego, CA, 92093

Differential algebraic equations (DAEs) arise in a variety of practical engineering systems. This work presents an open-source framework to numerically obtain the solution to DAEs, and further obtain the gradient of functions of interest with respect to design variables. A time-domain forward solver based on the stable backward difference formula (BDF) is implemented. The gradients are efficiently obtained using the adjoint method. Prior work has shown that the computation of six partial derivatives are sufficient to obtain the solution of both the forward and adjoint problems. In this work, the framework is implemented using the computational system design language (CSDL). CSDL enables the user to write high-level python code and automates the computation of the required derivatives. Thus, the framework saves time for users who are prototyping DAE formulations and solving complex engineering problems. The framework is validated against three different benchmark problems: spring-mass-damper system, pendulum system, and RLC circuit. The gradient from the adjoint solver is compared to that calculated using the finite difference method. It is found that the error of the both the forward and adjoint solutions is negligible. The implementation of benchmark problems also serves as examples of how users can formulate their own DAE systems in the open-source framework.

I. Introduction

Differential-algebraic equations (DAEs) are an important class of problems with wide applications in engineering. Consider finding an unknown vector-valued function $x(t)$, also called the independent variable or the state variable. $x(t)$ is a function of time t, and its derivative is denoted by $\dot{x}(t)$. Further, consider a vector-valued dependent variable $u(t)$. An algebraic equation is defined as one in which the derivatives of the variables are not present. A differential equation on the other hand is one in which the derivatives are present. A DAE system is a type of differential equation where one or more derivatives of the dependent variable $u(t)$ does not appear in the equations [1]. A general DAE system written in fully implicit form is

$$R(t, x(t), \dot{x}(t), u(t)) = 0 \tag{1}$$

The system is generally constrained by boundary and initial conditions. Compared to the ordinary differential equation (ODE), DAEs have more flexibility by treating the derivative of the state variable $\dot{x}$ as a first order variable, which is able to solve problems when $\dot{x}$ is unknown or singular. Solving DAEs to obtain the time-domain response is termed as the *forward solution*. Due to its flexibility, the DAEs are widely used in science and engineering for simulating dynamical behavior, including but not limited to electrical circuit systems [2], mechanical multi-body systems [3], structural systems [4], and fluid systems [5].

Numerical optimization techniques enable practitioners to virtually design complex engineering systems represented by DAEs with hundreds of variables and constraints. Polak [6] demonstrated that the basic property of optimization algorithms is convergence, which is generally solved through iterations with gradients as the search direction. Thus, an efficient and accurate computation of the gradient of a function of interest with respect to design variables is essential. The adjoint method is an efficient way of computing the gradients for optimization problems [7]. Obtained the gradient using the adjoint method is termed as the *adjoint solution*.

An approach to solving general DAEs was shown by Solano et al. [8]. A stable 2nd order backwards difference formula (BDF) was used to obtain the forward solution. The adjoint for the BDF systen was derived to obtain the gradient of a function of interest $f(\mu)$ with respect to the design variable vector μ. They demonstrated that any general

*Undergraduate Researcher, Department of Mechanical and Aerospace Engineering, 92093, AIAA Student Member
†Postdoctoral Scholar, Department of Mechanical and Aerospace Engineering, 92093, AIAA Member
‡Assistant Professor, Department of Mechanical and Aerospace Engineering, 92093, AIAA Member

DAE system can be solved if six partial derivative terms can be computed (Section II.C). The approach was implemented in a framework using a hybrid method of symbolic differentiation and automatic differentiation (AD) in the CasADi package [9] that automated the computation of the six derivatives.

A limitation of the approach by Solano et al. [8] is the implementation as a standalone package that is not well-suited to integrate into a comprehensive multidisciplinary design optimization (MDO) framework where coupled derivatives must be *automatically* computed.

A new tool named Computational System Design Language (CSDL) is under development at the Large Scale Design Optimization (LSDO) lab at the University of California, San Diego. CSDL automates derivative computation, both at the level of individual operations, and across tightly coupled disciplines, making it suitable for solving optimization problems using gradient-based approaches. The CSDL framework interfaces with OpenMDAO [10], which is a open-source software framework funded by NASA for multidisciplinary design optimization (MDO).

A package name OZONE has been developed in CSDL to simulate Ordinary Differential Equations (ODE), and obtain the gradient of ODE systems [7]. The motivation of this work is to develop an equivalent open-source simulation and gradient computation capability for DAE systems*.

A general forward solver using the BDF scheme is implemented in CSDL. The solver only requires a CSDL model representing the DAE system to be provided to it, and returns the time-domain response of the system. Further, given the DAE CSDL model, a CSDL model representing the function of the interest, the forward solution, and the design varaible(s) of interest; an adjoint solver is implemented that computes the gradient of the function of interest with respect to the design variables.

The framework is validated against three benchmark problems: spring-mass-damper system, pendulum system, and RLC circuit. As an example, consider the spring-mass-damper system. With user provided mass, stiffness, damping coefficient, initial position, and initial velocity of the object attached to the spring; the forward solver computes the position, velocity, and acceleration of the object as a function of time. Defining kinetic energy as the function of interest and the spring stiffness as the design variable, the derivative of the kinetic energy with respect to the spring stiffness is obtained. This derivative is compared to that obtained using the finite difference method. It is found that the framework accurately obtains the time-domain response and the gradient with negligible error.

The remainder of the paper is organized as follows. Section II provides background information on CSDL, OZONE, and the adjoint method for DAE systems. Section II applies the formulation to the three benchmark problems and validates the results. Finally, Section IV concludes the paper and indicates avenues for future research.

II. Background

A. Computational System Design Language

CSDL is an embedded domain specific language designed for solving MDO problems. CSDL enables engineers to operate at a high level of abstraction, without the need to implement low level algorithms, including derivative computation. CSDL automates derivative computation, both at the level of individual operations, and across tightly coupled disciplines, making it suitable for solving optimization problems using gradient-based approaches.

CSDL simulations are written, compiled, and run within a Python script. That is, the CSDL compiler translates Python code representing model behavior to low level implementation code used to simulate the model, and users have access to the executable object via a Python object. The CSDL compiler is a three stage compiler. Three stage compilers are split between a front end, which generates an intermediate representation of the code, a middle end, which performs implementation-independent optimizations on the generated intermediate representation, and a back end, which generates executable code. The intermediate representation of the model stores dependency relationships between variables and operations in the model. Different formulations such as forward or reverse mode differentiation can be implemented depending on the structure of the intermediate representation and the executable can be created from any language. The CSDL compiler's front end and middle end are implemented in one Python package and the back end (code generator) is implemented in a separate Python package. The separation between the code generator and the rest of the compiler enables users to select from a code generator based on performance and memory requirements for simulating the model without the need to modify model code.

*The open-source DAE solver can be found on GitHub. The repository is named DaeSystemsSolver

B. OZONE ODE Framework

The OZONE framework is an ODE solver implemented in the CSDL framework [7]. The general form of an ODE is

$$\frac{\partial x}{\partial t} = f(x, u) \tag{2}$$

where x is the state variable of the ODE, t is the time variable, u is the relevant parameters, and f is the ODE function. The ODE function f can be solved by integrating the function over time. In this case, three basic types of inputs are needed: state initial condition ($x(t = 0) = x_0$), timespan, and parameters. The logistics of OZONE follow this time-marching process in general.

To work with OZONE, the user needs to define the ODE function $\frac{\partial x}{\partial t} = f(x, u)$. The ODE function needs to be constructed in the CSDL model, where the inputs are x, u and the output is $\frac{\partial x}{\partial t}$. The inputs and the constraints on the outputs should be given, and the solver model should be inserted into the outer CSDL model to connect to other variables. In this work, we will follow similar implementation principles of OZONE to develop a framework for DAE systems.

C. Adjoint Method for DAE Systems

As we introduced in Section I, DAEs have more flexibility than the ODEs for solving problems. In fact, explicit ODEs are special cases of DAEs. The general form of the DAE is

$$R(t, x(t), \dot{x}(t), u(t)) = 0_{\mathbb{R}^{n_x}}, t \in [t_0, t_f] \subseteq \mathbb{R} \tag{3}$$

When the Jacobian $\frac{\partial R}{\partial \dot{x}}$ is non-singular, the ODE form can be derived as $\dot{x}(t) = f(t, x(t), u(t))$[3].

In this work we will use a stable 2nd order backward difference formula (BDF) scheme to integrate the DAE forward in time:

$$\dot{x}^t = \alpha_0 x^t + \alpha_1 x^{t-1} + \alpha_2 x^{t-2} \tag{4}$$

where $\alpha_0 = \frac{1.5}{\Delta t}$, $\alpha_1 = \frac{-2}{\Delta t}$, and $\alpha_2 = \frac{0.5}{\Delta t}$. Newton's method can be applied to solve the nonlinear DAE system at each time step, and has a form

$$\left[\frac{\partial R}{\partial x^t} + \alpha_0 \frac{\partial R}{\partial \dot{x}^t}\right] \delta_x = -r^t \tag{5}$$

where $\frac{\partial R}{\partial x^t}$ and $\frac{\partial R}{\partial \dot{x}^t}$ are Jacobians of the DAE system. The δ_x will be added to the last state and generate a new guess

$$x^{t+1} = x^t + \delta_x \tag{6}$$

As seen above, in order to do the time-marching, the Jacobians need to be computed at each step [8]. In this work, the DAE system will be written in CSDL. CSDL automates the task of computing the derivatives and thus will provide the necessary Jacobians to the time-marching solver.

In the context of design through numerical optimization, an objective/constraint functional can be expressed as a sum over time as

$$\begin{aligned} f(\boldsymbol{\mu}) &= \int_0^T F(\boldsymbol{\mu},\ \mathbf{x},\ \dot{\mathbf{x}},\ t)dt \\ &\approx \sum_{t=0}^{T} \Delta t F^t(\boldsymbol{\mu},\ \mathbf{x}^t,\ \dot{\mathbf{x}}^t) \end{aligned} \tag{7}$$

Here, the integrand F can be any arbitrary function of the state and design variables. The adjoint system at each time step is the following linear system, where the adjoint variable λ^t is solved

$$\begin{aligned} \left(\frac{\partial \mathbf{R}^t}{\partial \mathbf{x}} + \alpha_0 \frac{\partial \mathbf{R}^t}{\partial \dot{\mathbf{x}}}\right)^T \boldsymbol{\lambda}^t = -\left(\frac{\partial F^t}{\partial \mathbf{x}} + \alpha_0 \frac{\partial F^t}{\partial \dot{\mathbf{x}}}\right) - \\ \left(\sum_{p=1}^{P} \alpha_p \frac{\partial \mathbf{R}^{t+p}}{\partial \dot{\mathbf{x}}}^T \boldsymbol{\lambda}_{t+p} - \sum_{p=1}^{P} \alpha_p \frac{\partial F^{t+p}}{\partial \dot{\mathbf{x}}}^T\right) \end{aligned} \tag{8}$$

Once the adjoint variables at each time step are computed, the derivative of the functional of interest (FoI) w.r.t. design variables can be found as

$$\frac{df}{d\boldsymbol{\mu}} = \sum_{t=0}^{T} \Delta t \frac{\partial F^t}{\partial \boldsymbol{\mu}} + \sum_{t=0}^{T} \Delta t \left(\boldsymbol{\lambda}^t\right)^T \frac{\partial \mathbf{R}^t}{\partial \boldsymbol{\mu}} \tag{9}$$

The formulation presented here results in a general method to obtain the gradient for a system of residual equations time-marched using a BDF scheme. The quantities required in Eq. 8 and Eq. 9 are:

- $\frac{\partial \mathbf{R}}{\partial \mathbf{x}}$- the Jacobian matrix of the derivative of the residual equations w.r.t. the state variables
- $\frac{\partial \mathbf{R}}{\partial \dot{\mathbf{x}}}$- the Jacobian matrix of the derivative of the residual equations w.r.t. the first time derivative of the state variables
- $\frac{\partial F}{\partial \mathbf{x}}$- the vector of derivatives of the function of interest w.r.t. the state variables
- $\frac{\partial F}{\partial \dot{\mathbf{x}}}$- the vector of derivatives of the function of interest w.r.t. the first time derivative of the state variables

The gradient is then computed using Eq. 9 for which the following two additional quantities are required:

- $\frac{\partial F}{\partial \boldsymbol{\mu}}$- the vector of derivatives of the function of interest w.r.t. the design variables
- $\frac{\partial \mathbf{R}}{\partial \boldsymbol{\mu}}$- the matrix of derivatives of the residual equations w.r.t. the design variables

III. Validation: Benchmark Problems

Three benchmark problems are implemented in order to validate the effectiveness of our time-marching solver for DAE system and the adjoint solver for gradient. The resulting gradients from the adjoint solver are compared with the those resulting from finite difference method. The computation process of the spring-mass-damper system will be explained in detail as an example to demonstrate the mathematical logistics.

A. Spring-Mass-Damper System

A simple spring-mass-damper system is shown in Fig. 1(a), and its free body diagram is shown in Fig. 1(b) [11]. From which, the force equation $F = -(kx + b\dot{x})$ can be derived, where k is the stiffness of the spring, and b is the damping coefficient of the damper. According to Newton's second law, $F = ma = m\ddot{x}$, we could further derive the equation $m\ddot{x} = -(kx + b\dot{x})$, where m is the mass of the object attached to the spring.

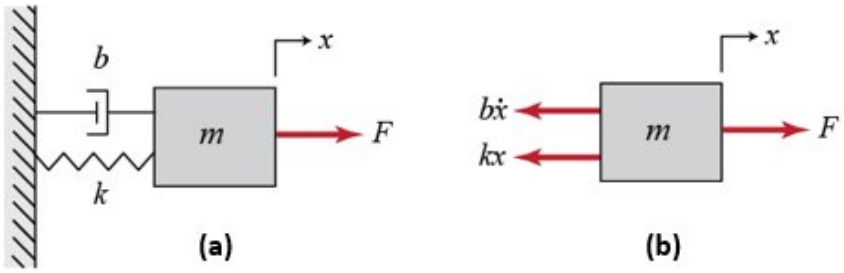

Fig. 1 Free body diagram of spring-mass-damper system

The state variable $\mathbf{x} = \begin{pmatrix} x \\ \dot{x} \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ can be obtained introducing $x_1 = x$ and $x_2 = \dot{x}$, and thus $\dot{\mathbf{x}} = \begin{pmatrix} \dot{x} \\ \ddot{x} \end{pmatrix} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix}$. Implementing the state variable to the derived equations, the DAE system can be constructed:

$$\mathbf{R}(\mathbf{x}, \dot{\mathbf{x}}, t, m, b, k) = \begin{pmatrix} m\dot{x}_2 + bx_2 + kx_1 \\ x_2 - \dot{x}_1 \end{pmatrix} = 0 \tag{10}$$

Thus, the Jacobian matrix $\frac{\partial \mathbf{R}}{\partial \mathbf{x}}$ is

$$\frac{\partial \mathbf{R}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial R_1}{\partial x_1} & \frac{\partial R_1}{\partial x_2} \\ \frac{\partial R_2}{\partial x_1} & \frac{\partial R_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} k & b \\ 0 & 1 \end{pmatrix} \tag{11}$$

and the Jacobian matrix $\frac{\partial \mathbf{R}}{\partial \dot{\mathbf{x}}}$ is

$$\frac{\partial \mathbf{R}}{\partial \dot{\mathbf{x}}} = \begin{pmatrix} \frac{\partial R_1}{\partial \dot{x}_1} & \frac{\partial R_1}{\partial \dot{x}_2} \\ \frac{\partial R_2}{\partial \dot{x}_1} & \frac{\partial R_2}{\partial \dot{x}_2} \end{pmatrix} = \begin{pmatrix} 0 & m \\ -1 & 0 \end{pmatrix} \tag{12}$$

Using the BDF scheme with a general form of $\dot{x}^t = \alpha_0 x^t + \alpha_1 x^{t-1} + \alpha_2 x^{t-2}$, the DAE system can be solve by Newton's method. To solve δ_x to get the next time step, plug Eq. 11 and Eq. 12 into Eq. 5

$$\left[\frac{\partial R}{\partial x^t} + \alpha_0 \frac{\partial R}{\partial \dot{x}^t} \right] \delta_x = \left[\begin{pmatrix} k & b \\ 0 & 1 \end{pmatrix} + \frac{1.5}{\Delta t} \begin{pmatrix} 0 & m \\ -1 & 0 \end{pmatrix} \right] \delta_x = -r \tag{13}$$

The input values for m, b, and k are 2.5, 0.2, and 5 respectively. Using Newton's method, this algorithm converges the residual r to zero as the iteration continues. The result from the time-marching solver is shown in figure 2.

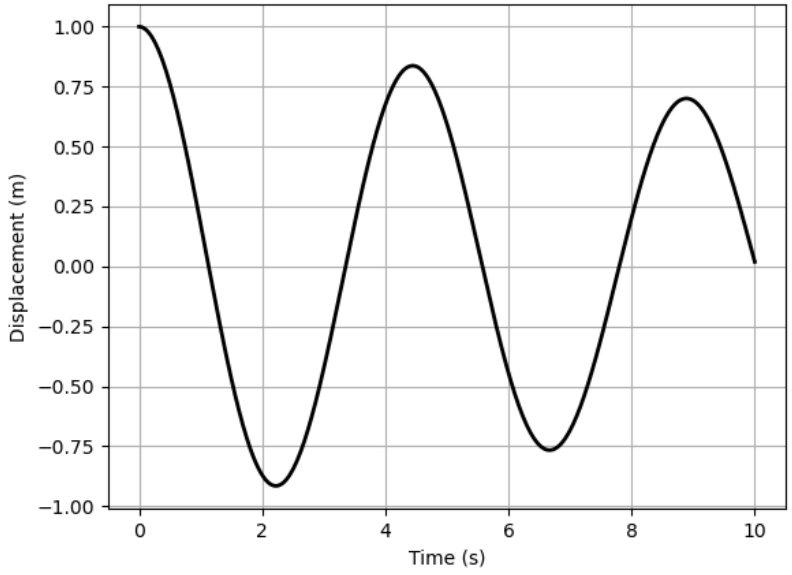

Fig. 2 Mass displacement as a function of time

With solved $\mathbf{x}$ and $\dot{\mathbf{x}}$ at each time step, the gradient term $\frac{\partial R^t}{\partial \boldsymbol{\mu}}$ can be obtained

$$\frac{\partial R^t}{\partial \boldsymbol{\mu}} = \begin{pmatrix} \frac{\partial R_1^t}{\partial m} & \frac{\partial R_1^t}{\partial b} & \frac{\partial R_1^t}{\partial k} \\ \frac{\partial R_2^t}{\partial m} & \frac{\partial R_2^t}{\partial b} & \frac{\partial R_2^t}{\partial k} \end{pmatrix} = \begin{pmatrix} \dot{x_2} & x_2 & x_1 \\ 0 & 0 & 0 \end{pmatrix} \tag{14}$$

At this point, $\frac{\partial \mathbf{R}}{\partial \mathbf{x}}$, $\frac{\partial \mathbf{R}}{\partial \dot{\mathbf{x}}}$, and $\frac{\partial \mathbf{R}}{\partial \boldsymbol{\mu}}$ of the six required quantities are derived. In order to get $\frac{\partial F}{\partial \mathbf{x}}$, $\frac{\partial F}{\partial \dot{\mathbf{x}}}$, and $\frac{\partial F}{\partial \boldsymbol{\mu}}$, a function of interest has to be decided, depending which property the user desires to optimize. In this example, the function of interest is based on the elastic potential energy of the spring.

$$f = \sum_{i=1}^{n} \Delta t \frac{1}{2} k x_{1_t}^2 = \sum_{i=1}^{n} F_i \tag{15}$$

$\frac{\partial F}{\partial \mathbf{x}}$, $\frac{\partial F}{\partial \dot{\mathbf{x}}}$, and $\frac{\partial F}{\partial \boldsymbol{\mu}}$ are respectively

$$\frac{\partial F}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial F}{\partial x_1} & \frac{\partial F}{\partial x_2} \end{pmatrix} = \begin{pmatrix} \Delta t k x_1 & 0 \end{pmatrix} \tag{16}$$

$$\frac{\partial F}{\partial \dot{\mathbf{x}}} = \begin{pmatrix} \frac{\partial F}{\partial \dot{x}_1} & \frac{\partial F}{\partial \dot{x}_2} \end{pmatrix} = \begin{pmatrix} 0 & 0 \end{pmatrix} \tag{17}$$

$$\frac{\partial F^t}{\partial \boldsymbol{\mu}} = \begin{pmatrix} \frac{\partial F^t}{\partial m} & \frac{\partial F^t}{\partial b} & \frac{\partial F^t}{\partial k} \end{pmatrix} = \begin{pmatrix} 0 & 0 & \frac{1}{2}\Delta t x_1^2 \end{pmatrix} \tag{18}$$

To compute adjoint variables from the adjoint equation at each time step, apply Eq. 11, Eq. 12, Eq. 16, and Eq.17 to Eq. 8

$$\left[\begin{pmatrix} k & b \\ 0 & 1 \end{pmatrix} + \alpha_0 \begin{pmatrix} 0 & m \\ -1 & 0 \end{pmatrix} \right]^T \lambda^t = - \begin{pmatrix} \Delta t k x_1 \\ 0 \end{pmatrix} - \sum_{p=1}^{2} \alpha_p \begin{pmatrix} 0 & m \\ -1 & 0 \end{pmatrix}^T \lambda^{t+p} \tag{19}$$

With computed adjoint variables, the derivative of function of interest with respect to design variables can be given by Eq. 9

$$\begin{pmatrix} \frac{\partial f}{\partial m} & \frac{\partial f}{\partial b} & \frac{\partial f}{\partial k} \end{pmatrix} = \sum_{t=0}^{T} \Delta t \begin{pmatrix} 0 & 0 & \frac{1}{2}\Delta t x_{1_t}^2 \end{pmatrix} + \sum_{t=0}^{T} \Delta t (\lambda^t)^T \begin{pmatrix} \dot{x_2} & x_2 & x_1 \\ 0 & 0 & 0 \end{pmatrix} \tag{20}$$

In this example, only $\frac{\partial f}{\partial k}$ is relevant to the function of interest. Using adjoint method, the gradient of function of interest with respect to design variable k is **1.1605794**.

For the purpose of verifying results from the adjoint method, the gradients are again computed by finite difference method. Implementing forward difference formula $f'(a) \approx \frac{f(a+h)-f(a)}{h}$, specifically

$$f'(m, b, k) = f'(m, b, k + \delta) \tag{21}$$

$$\approx \frac{f(m, b, k + \delta) - f(m, b, k)}{\delta} \tag{22}$$

where δ is a small perturbation of the design variable k, the gradient computed from the finite difference method is **1.1608199**. The percentage difference between the two resulting gradients is **0.0207**%.

B. Pendulum Mechanics

Another classic mechanical model is the pendulum system. In this example, the adjoint solver is applied to a pendulum system with induced damper factor b. Similar to the spring-mass-damper system, $m\ddot{s} = -(mg\sin(\theta) + b\dot{s})$ can be derived from applying Newton's second law, where s is the arc length travelled by the mass. Set L be the length of the pendulum. Then, the length of the arc is $s = L\theta$, so $\dot{s} = L\dot{\theta}$, and $\ddot{s} = L\ddot{\theta}$. Thus, the equation $\ddot{\theta} + \frac{b}{m}\dot{\theta} + \frac{g}{L}\sin\theta = 0$ is obtained.

Set up the state variable with $x = \begin{pmatrix} \theta \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ and the first order time derivative of the state variable $\dot{x} = \begin{pmatrix} \dot{\theta} \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix}$, the DAE system can be constructed with the derived equation

$$\mathbf{R}(x, \dot{x}, t, L, b, m) = \begin{pmatrix} \dot{x}_2 + \frac{b}{m}x_2 + \frac{g}{L}\sin x_1 \\ x_2 - \dot{x}_1 \end{pmatrix} = 0 \tag{23}$$

The input values for L, b, and m are 1, 0.5, 1 respectively[†]. Using BDF scheme and newton's method as explained in the previous problem, the resulting state variables of the DAE system are shown in figure 3.

The function of interest in this problem is decided by the kinetic energy $K = \frac{1}{2}mv^2$ of the mass m, where $v = L\dot{\theta}$

$$f = \sum_{i=1}^{n} \Delta t \frac{1}{2} m (L x_2)^2 = \sum_{i=1}^{n} F_i \tag{24}$$

Using adjoint method, the final gradient of this function of interest with respect to design variable L is **8.8963550**. For the finite difference method, again apply forward difference formula

$$f'(L, b, m) \approx \frac{f(L + \delta, b, m) - f(L, b, m)}{\delta} \tag{25}$$

which gives a gradient of **8.8969986**. The percentage difference between the results of the two methods is **0.0072**%.

[†]The parameters for the pendulum system are obtained from the following webpage: `https://skill-lync.com/student-projects/Simulation-of-a-Simple-Pendulum-on-Python-95518`

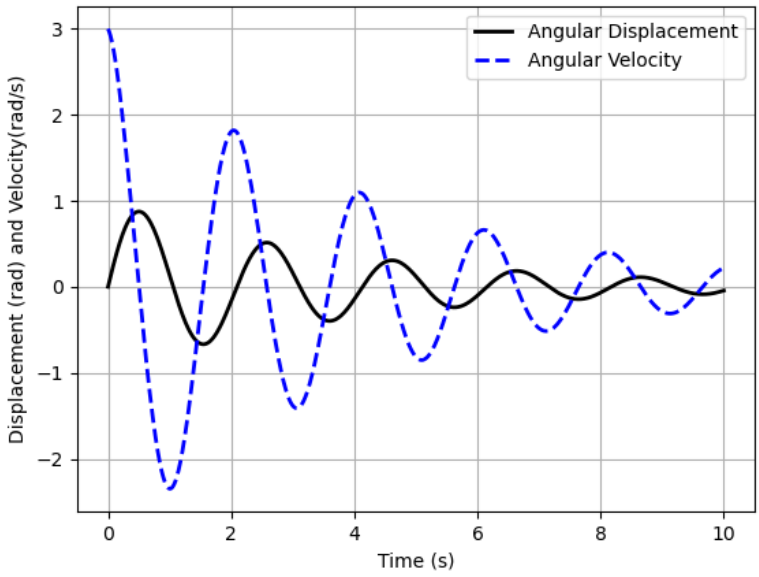

Fig. 3 Angular displacement and angular velocity of the pendulum as a function of time

C. RLC Circuit

According to Kirchhoff's Law, $V_R + V_L + V_C = V$, where V_R, V_L, V_C are voltages corresponding to resistor, inductor, and capacitor in the circuit; V is the constant voltage input of the system. Further, the voltages can be computed with $V_R = RI(t)$ (where I is current, and R is resistance of the resistor), $V_L = L\frac{dI(t)}{dt}$(there L is the inductance), and $V_C = V(0) + \frac{1}{C}\int_0^t I(\tau)\,d\tau$ (where C is the capacitance) [12].

Set up the state variable $x = \begin{pmatrix} I \\ V_L \\ V_C \\ V_R \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$ and the first order time derivative of the state variable $\dot{x} = \begin{pmatrix} \dot{I} \\ \dot{V}_L \\ \dot{V}_C \\ \dot{V}_R \end{pmatrix} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix}$.

The DAE system can be constructed as shown in Musau et al. [13]

$$\mathbf{R}(x, \dot{x}, t, R, L, C, V) = \begin{pmatrix} x_2 - L\dot{x}_1 \\ \frac{1}{C}x_1 - \dot{x}_3 \\ \dot{x}_4 - Rx_1 \\ x_2 + x_3 + x_4 - V \end{pmatrix} = 0 \tag{26}$$

The input values for R, L, C and V are 1.1, 1.6, 0.8, 2.4 respectively[‡]. Again, applying BDF scheme and newton's method, the computed results of the DAE system are shown in figure 4.
The function of interest in this example is chosen to be the power of the circuit $P = VI$, which gives

$$f = \sum_{i=1}^{n} \Delta t V x_1 = \sum_{i=1}^{n} F_i \tag{27}$$

The gradient of our function of interest with respect to design variable V using adjoint method is **−0.2979395**. The graient from the finite difference method is **−0.2978997**. The percentage difference between the results from adjoint method and finite difference method is **0.0133**%.

[‡]The parameters for the RLC circuit are obtained from the following webpage: `https://2021.help.altair.com/2021/compose/personal/en_us/topics/compose/stc_solving_diff_alg_equ_t.htm`

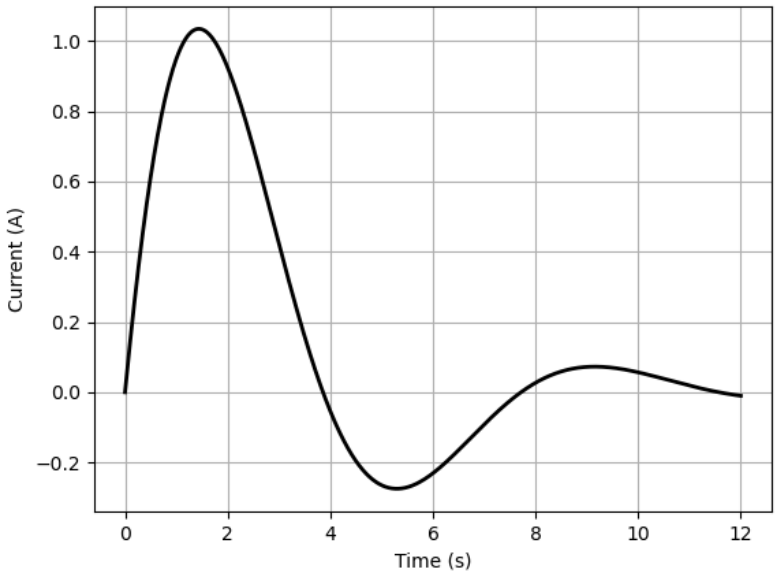

Fig. 4 Current in the RLC circuit as a function of time

IV. Conclusion

This work presents a framework to solve DAE systems. The framework obtains the time-domain response of the DAE system using the BDF scheme, and the gradient of functions of interest with respect to design variables using the adjoint method.

The novelty of the framework is the use of CSDL that enables the user to write high-level python code and automates the computation of the required derivatives. Users need only provide a CSDL model of the DAE and the FoI instead of deriving the gradient themselves or providing complicated function handles. Thus, the framework saves time for users who are prototyping DAE formulations and solving complex engineering problems.

The framework is validated against three benchmark problems: spring-mass-damper-system, pendulum system, and RLC circuit; each with different functions of interest and design variables. It is observed that the difference between the gradients from the adjoint method and the finite difference method is negligible.

While this work demonstrates the framework on the benchmark problems, the framework is applicable to any general DAE system. For example, aircraft wings can be represented as nonlinear beam elements. This leads to a DAE system with 18 states per node in the beam [4]. It has been shown that such a system can efficiently size structural components of the aircraft wing subjected to dynamic loads specified by 14-CFR regulations.

The benchmark problems also serve as illustrative examples on how to implement DAE systems as CSDL models, and how to use the forward and adjoint solvers. The implementations are available open-source on GitHub. Future work will integrate this work with OZONE to create a comprehensive tool to solve ODEs and DAEs, and can be used in large-scale MDO.

Acknowledgments

The authors acknowledge the help from Mark Sperry on implementing the models in CSDL.

References

[1] Kunkel, P., and Mehrmann, V., *Differential-algebraic equations: analysis and numerical solution*, Vol. 2, European Mathematical Society, 2006.

[2] Estévez Schwarz, D., and Tischendorf, C., "Structural analysis of electric circuits and consequences for MNA," *International*

Journal of Circuit Theory and Applications, Vol. 28, No. 2, 2000, pp. 131–162. https://doi.org/https://doi.org/10.1002/(SICI)1097-007X(200003/04)28:2<131::AID-CTA100>3.0.CO;2-W.

[3] Gerdts, M., *Optimal control of ODEs and DAEs*, Walter de Gruyter, 2011.

[4] Sarojini, D., and Mavris, D., "Structural Analysis and Optimization of Wings Subjected to Dynamic Loads," *AIAA Journal*, Vol. 60, No. 2, 2022, pp. 1013–1023. https://doi.org/https://doi.org/10.2514/1.J060931.

[5] John, V., Matthies, G., and Rang, J., "A comparison of time-discretization/linearization approaches for the incompressible Navier–Stokes equations," *Computer Methods in Applied Mechanics and Engineering*, Vol. 195, No. 44-47, 2006, pp. 5995–6010. https://doi.org/https://doi.org/10.1016/j.cma.2005.10.007.

[6] Polak, E., "Algorithms and consistent approximations," *Optimization*, 1997.

[7] Hwang, J. T., and Munster, D., "Solution of ordinary differential equations in gradient-based multidisciplinary design optimization," *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2018, p. 1646. https://doi.org/https://doi.org/10.2514/6.2018-1646.

[8] Solano, D., Sarojini, D., Rajaram, D., and Mavris, D. N., "Adjoint-based analysis and optimization of beam-like structures subjected to dynamic loads," *Structural and Multidisciplinary Optimization*, Vol. 65, No. 2, 2022, pp. 1–19. https://doi.org/https://doi.org/10.1007/s00158-021-03141-5.

[9] Andersson, J. A., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M., "CasADi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, Vol. 11, No. 1, 2019, pp. 1–36. https://doi.org/https://doi.org/10.1007/s12532-018-0139-4.

[10] Heath, C., and Gray, J., "OpenMDAO: framework for flexible multidisciplinary design, analysis and optimization methods," *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 20th AIAA/ASME/AHS Adaptive Structures Conference 14th AIAA*, 2012, p. 1673. https://doi.org/https://doi.org/10.2514/6.2012-1673.

[11] Hamdan, S., Oztop, E., and Ugurlu, B., "Force Reference Extraction via Human Interaction for a Robotic Polishing Task: Force-Induced Motion," *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2019, pp. 4019–4024. https://doi.org/10.1109/SMC.2019.8914009.

[12] Irwin, J. D., and Nelms, R. M., *Basic engineering circuit analysis*, John Wiley & Sons, 2020.

[13] Musau, P., Lopez, D. M., Tran, H.-D., and Johnson, T. T., "Linear Differential-Algebraic Equations (Benchmark Proposal)," *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems*, EPiC Series in Computing, Vol. 54, edited by G. Frehse, EasyChair, 2018, pp. 174–184. https://doi.org/10.29007/4gj7, URL https://easychair.org/publications/paper/mc5T.