Journal of The Electrochemical Society

OPEN ACCESS

Implementation of Dual Number Automatic Differentiation with John Newman's BAND Algorithm

Journal of The Electrochemical Society, 2021 **168** 113501

Implementation of Dual Number Automatic Differentiation with John Newman's BAND Algorithm

Nicholas W. Brady,[1,2,3,*,z] Maarten Mees,[3,4,5] Philippe M. Vereecken,[3,4,5,*] and Mohammadhosein Safari[1,2,3]

[1]*Institute for Materials Research (IMO-imomec), UHasselt, 3500 Hasselt, Belgium*
[2]*IMEC division IMOMEC, 3590 Diepenbeek, Belgium*
[3]*Energyville, 3600 Genk, Belgium*
[4]*Imec, 3001 Leuven, Belgium*
[5]*cmacS, M2S, KU-Leuven, 3001 Leuven, Belgium*

This paper asserts that the development of continuum-scale mathematical models utilizing John Newman's BAND subroutine can be simplified through the use of dual number automatic differentiation. This paper covers the salient features of the BAND algorithm as well as dual numbers and how they can be leveraged to algorithmically linearize systems of partial differential equations; these two concepts can be combined to produce accurate and computationally efficient models while significantly reducing the amount of personnel time necessary by eliminating the time-consuming process of equation linearization. As a result, this methodology facilitates more rapid model prototyping and establishes a more intuitive relationship between the numerical model and the differential equations. By utilizing an existing and validated programming module, dnadmod, these advantages are achieved without burdening the general user with significant additional programming overhead.
 [DOI: 10.1149/1945-7111/ac3274]

Manuscript submitted September 3, 2021; revised manuscript received October 15, 2021. Published November 8, 2021.

Driven by an increasing number of researchers, technological advancements, and the ability to cheaply store large amounts of digital information, experimental data is abundant. Physics-based mathematical modeling, in conjunction with experimental data, is essential to scientific discovery and technological advancement because it allows one to quantitatively test hypotheses, measure physical parameters, and gives the necessary precision to perform quantitative design, optimization, and control—which drive the maturization of a technology. This is true in lithium-ion battery applications where highly coupled physical process—mechanical factors, thermodynamics, thermal effects, electrodynamics, and chemical reactions—all need to be understood to fully optimize and control performance. Because model development is a time-consuming process, it can be difficult for it to keep pace with the rate of experimental data collection, which can stifle innovation.

Computers and software may provide the solution. Computers are ubiquitous and the costs of computational resources continue to drop exponentially.[1,2] In addition, the low costs of computational resources has led to huge advancements in the fields of machine learning and artificial intelligence and these techniques are widely accessible even to non-experts. With cheap computers and sophisticated software, it is becoming increasingly the case that the rate-limiting (and most costly) steps are those that involve human input.

To elucidate where possible bottle-necks exist, it is helpful to segregate the model development and implementation process into distinct steps:

1. Developing a physical hypothesis
2. Developing a mathematical model to emulate the physics
3. Developing a numerical program from the mathematical expressions
4. Performing quantitative parameter estimation
5. Performing quantitative optimization, design, and control

Although advanced computational techniques are routinely linked with physics-based models in the scientific and battery literature to perform tasks such as physical parameter estimation[3] and optimization (steps 4 and 5 above), there is a need to go further and move as much of step 3, sometimes called discretization, from a human task to a computer task. Using dual number automatic differentiation, one can algorithmically perform equation linearization, significantly decreasing the amount of human input needed in developing a numerical program.

While the use of dual number automatic differentiation has been validated in computational fluid dynamics (CFD),[4–7] to the authors' knowledge this technique is not widely used in modeling electrochemistry or battery applications. This paper leverages existing and validated tools: BAND,[8] material balances based on a control volume approach,[9] and dnadmod[4] to develop a systematic approach to modeling physical conditions relevant to electrochemical and battery applications. This approach is accurate to machine precision, computationally efficient, physically intuitive, and most significantly, eliminates the need for user-input to the linearization process.

This paper illustrates how paradigms currently used in battery model development can be easily integrated with dual number automatic differentiation to build a model development process that is both more physically intuitive while also eliminating the need for human interaction with equation linearization. This paper first reviews the salient features of BAND and the control volume approach, as well as commonly used techniques for equation linearization, and the mathematical concept of a dual number. Finally, there is a detailed description of how dual number automatic differentiation can be coupled with the control volume approach and BAND to systematically solve systems of coupled differential equations, while minimizing the need for human input. The appendix provides selections of Fortran code that the reader is invited to copy and use to model systems relevant to their own research.

The State of the Art

This Section gives a summary of the BAND algorithm, the control volume or finite element formulation, methods of equation linearization, as well as the properties of dual numbers. This Section does not contain any novel developments but lays the groundwork for understanding the utility of dual number automatic differentiation and how it can be used to simplify the programming necessary to solve problems with BAND.

*Electrochemical Society Member.
[z]E-mail: nwb2112@columbia.edu

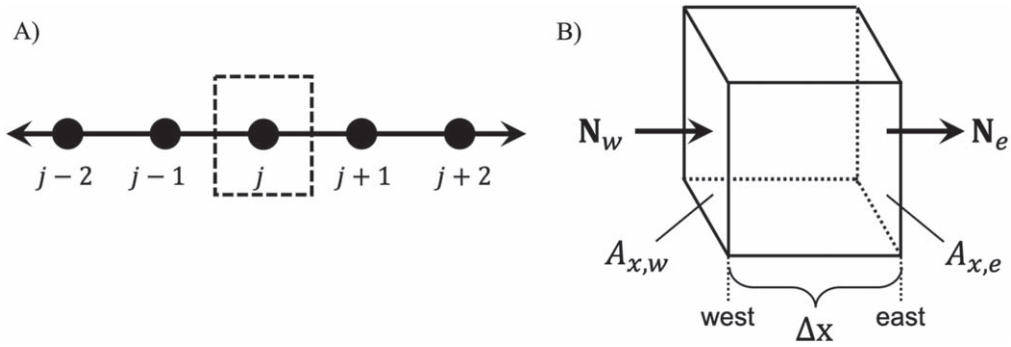

Figure 1. (A) Diagram of the discretization process and (B) a cartoon representation of the control volume approach. The "western" interface lies between nodes $j-1$ and j, while the "eastern" interface lies between j and $j+1$. The cross-sectional areas at these respective interfaces are indicated as $A_{x,w}$ and $A_{x,e}$. The distance between adjacent nodes is marked as Δx, while the 3-dimensional space contained within the control volume is referred to as ΔV. $\mathbf{N}_w$ and $\mathbf{N}_e$ represent the fluxes through the western and eastern interfaces, respectively.

BAND.—John Newman first published his BAND and MATINV subroutines in the 1960s.[8,10,11] These subroutines allowed for systems of differential equations to be iteratively solved with improved computational efficiency. To numerically solve systems of coupled differential equations, the equations need to be discretized over a set of node points, see Fig. 1A. The computational efficiency of Newman's BAND subroutine is derived from its assumption that the solution at node j only depends on the values of the variables at node j, and the adjacent nodes, $j-1$, $j+1$, thus creating a block tridiagonal matrix, which can be solved explicitly and efficiently. Detailed explanations of the mathematics and numerics behind BAND can be found in the literature,[8,12] but the salient features are summarized here.

BAND is essentially a multivariate Newton-Raphson method. Consider the equation

$$g(c) = 0 \qquad [1]$$

where g is an arbitrary continuous function that can be approximated using a Taylor series expansion

$$g(c) = g(c^\circ) + \left.\frac{dg}{dc}\right|_\circ (c - c^\circ) + \ldots \qquad [2]$$

where the quadratic, cubic, and higher order terms are ignored. Let $\Delta c = c - c^\circ$ and call this the change variable. Using the Taylor series approximation, the value of Δc that solves Eq. 1 is

$$\Delta c = -\frac{g(c^\circ)}{dg/dc|_\circ} \qquad [3]$$

One can iteratively solve this problem with quadratic convergence by successively updating the value of c°

$$c^\circ = c^\circ_{prev} + \Delta c \qquad [4]$$

where c°_{prev} is the value of c° from the previous iteration.

For a system of N governing equations, g_i, and N unknowns, C_k, where the equations have been discretized over NJ node points, as shown in Fig. 1, one can apply a multivariate Newton-Raphson approach at each node point j:

$$g_i(C_{k,j-1}, C_{k,j}, C_{k,j+1}) = 0 \qquad [5]$$

here i corresponds to the equation number (1 to N), k corresponds to the variable number (1 to N) and j the node index (1 to NJ). Note, this assumes that the solution to the governing equation at j only depends on the values of the variables at node j and the adjacent nodes, $j-1$, $j+1$. Using a multivariate Taylor series expansion yields the following equation

$$\sum_k A^\circ_{i,k}\Delta C_{k,j-1} + \sum_k B^\circ_{i,k}\Delta C_{k,j} + \sum_k D^\circ_{i,k}\Delta C_{k,j+1} = g^\circ_{i,j} \qquad [6]$$

where $g^\circ_{i,j}$ represents the evaluation of the governing equation i using the values of the variables from the previous iteration, i.e.

$$g^\circ_{i,j} = g_i(C^\circ_{k,j-1}, C^\circ_{k,j}, C^\circ_{k,j+1}) \qquad [7]$$

and the coefficients $A^\circ_{i,k}$, $B^\circ_{i,k}$, and $D^\circ_{i,k}$ are defined as

$$A^\circ_{i,k} = -\left.\frac{\partial g_{i,j}}{\partial C_{k,j-1}}\right|_\circ,\ B^\circ_{i,k} = -\left.\frac{\partial g_{i,j}}{\partial C_{k,j}}\right|_\circ,$$
$$D^\circ_{i,k} = -\left.\frac{\partial g_{i,j}}{\partial C_{k,j+1}}\right|_\circ,\ G^\circ_{i,j} = g^\circ_{i,j} \qquad [8]$$

The detailed mathematics of how to tractably and iteratively solve these coupled systems of equations is covered in Appendix C of *Electrochemical Systems*.[8] For this paper, the reader only needs to understand that the coefficients outlined in Eq. 8 are the partial derivatives of $g_{i,j}$. There are two significant challenges to working with BAND:

1. The coefficients $A^\circ_{i,k}$, $B^\circ_{i,k}$, $D^\circ_{i,k}$, $G^\circ_{i,j}$ do not intuitively relate to the differential forms of the equations commonly encountered in scientific and engineering situations
2. The efficiency of BAND derives from the block tridiagonal matrix structure as well as the assumption that the system is linear. This means that non-linear systems need to be linearized to be congruent with the program's structure. The linearization process, i.e. evaluating $\partial g_i/\partial C^\circ_{k,j}$, is not always trivial and until now, a general computerized process for linearization has not been introduced to be used with BAND.

Control volume formulation.—One approach to make BAND more intuitively accessible was introduced by J. Deliang Yang.[9] The strategy utilized a numerical technique called the finite volume method (FVM) or control volume approach;[12–17] a visual illustration of this approach is given in Fig. 1B and the strategy is summarized here. The control volume is centered at node j, with the "western" interface lying between nodes $j-1$ and j and the "eastern" interface lying between nodes j and $j+1$; the 3-dimensional space contained within the control volume is ΔV, and the linear distance from the "western" to "eastern" interface is Δx.

The shell balance over the control volume is written as:

$$\left\{\begin{array}{c}\text{Rate of}\\ \text{Accumulation}\end{array}\right\}=\{\text{Rate In}\} -\{\text{Rate Out}\} + \left\{\begin{array}{c}\text{Rate of}\\ \text{Generation}\end{array}\right\} \quad [9]$$

where the terms Rate In and Rate Out are related to transport phenomena, and Rate of Generation is related to the reaction expression. A general mathematical expression for the control volume indicated in Fig. 1B can be written as:

$$\Delta V(\text{Accum}_i) = A_{x,w}\mathbf{N}_{i,w} - A_{x,e}\mathbf{N}_{i,e} + \Delta V R_i \quad [10]$$

where Accum_i and R_i are the rates of accumulation and generation of specie i per unit volume, respectively, and $\mathbf{N}_{i,w}$ and $\mathbf{N}_{i,e}$ are the fluxes of specie i through the "western" and "eastern" faces, respectively. These expressions for flux, generation, and accumulation can each be linearized:

$$\mathbf{N}_{i,w} = \mathbf{N}^{\circ}_{i,w} + \sum_k \left.\frac{\partial \mathbf{N}_{i,w}}{\partial(\nabla c_{k,w})}\right|_{\circ} \Delta(\nabla c_{k,w}) + \sum_k \left.\frac{\partial \mathbf{N}_{i,w}}{\partial c_{k,w}}\right|_{\circ} \Delta c_{k,w} \quad [11]$$

$$\mathbf{N}_{i,e} = \mathbf{N}^{\circ}_{i,e} + \sum_k \left.\frac{\partial \mathbf{N}_{i,e}}{\partial(\nabla c_{k,e})}\right|_{\circ} \Delta(\nabla c_{k,e}) + \sum_k \left.\frac{\partial \mathbf{N}_{i,e}}{\partial c_{k,e}}\right|_{\circ} \Delta c_{k,e} \quad [12]$$

$$R_i = R_i^{\circ} + \sum_k \left.\frac{\partial R_i}{\partial c_k}\right|_{\circ} \Delta c_k \quad [13]$$

$$\text{Accum}_i = \sum_k \left.\frac{\partial(\text{Accum}_i)}{\partial(\Delta c_k)}\right|_{\circ} \Delta c_k; \quad \frac{\partial c_k}{\partial t} = \frac{\Delta c_k}{\Delta t} \quad [14]$$

where c_k is the average concentration of specie i within the control volume, $c_{k,w}$ and $c_{k,e}$ are the concentration of specie i at the "western" and "eastern" interfaces, respectively, and $\nabla c_{k,w}$ and $\nabla c_{k,e}$ are the spatial concentration gradients at these respective interfaces.

Inserting the expressions listed in Eqs. 11–14 into Eq. 10, one obtains the following expression:

$$\begin{aligned}-A_{x,w}\mathbf{N}^{\circ}_{i,w} + A_{x,e}\mathbf{N}^{\circ}_{i,e} - \Delta V R_i^{\circ} &= A_{x,w}\left[\sum_k \left.\frac{\partial \mathbf{N}_{i,w}}{\partial(\nabla c_{k,w})}\right|_{\circ} \Delta(\nabla c_{k,w}) + \sum_k \left.\frac{\partial \mathbf{N}_{i,w}}{\partial c_{k,w}}\right|_{\circ} \Delta c_{k,w}\right]\\ &- A_{x,e}\left[\sum_k \left.\frac{\partial \mathbf{N}_{i,e}}{\partial(\nabla c_{k,e})}\right|_{\circ} \Delta(\nabla c_{k,e}) + \sum_k \left.\frac{\partial \mathbf{N}_{i,e}}{\partial c_{k,e}}\right|_{\circ} \Delta c_{k,e}\right]\\ &+ \Delta V \sum_k \left[\left(\left.\frac{\partial R_i}{\partial c_k}\right|_{\circ} - \left.\frac{\partial(\text{Accum}_i)}{\partial(\Delta c_k)}\right|_{\circ}\right)\Delta c_k\right]\end{aligned} \quad [15]$$

from which the following terms can be defined:

$$d_{i,k}|_w = A_{x,w}\left.\frac{\partial \mathbf{N}_{i,w}}{\partial(\nabla c_{k,w})}\right|_{\circ}; \quad d_{i,k}|_e = A_{x,e}\left.\frac{\partial \mathbf{N}_{i,e}}{\partial(\nabla c_{k,e})}\right|_{\circ} \quad [16]$$

$$f_{i,k}|_w = A_{x,w}\left.\frac{\partial \mathbf{N}_{i,w}}{\partial c_{k,w}}\right|_{\circ}; \quad f_{i,k}|_e = A_{x,e}\left.\frac{\partial \mathbf{N}_{i,e}}{\partial c_{k,e}}\right|_{\circ} \quad [17]$$

$$r_{i,k} = \Delta V\left[\left.\frac{\partial R_i}{\partial c_k}\right|_{\circ} - \left.\frac{\partial(\text{Accum}_i)}{\partial(\Delta c_k)}\right|_{\circ}\right] \quad [18]$$

$$g^{\circ}_{i,j} = -(A_{x,w}\mathbf{N}^{\circ}_{i,w} - A_{x,e}\mathbf{N}^{\circ}_{i,e} + \Delta V R_i^{\circ}) \quad [19]$$

The coefficients outlined in Eqs. 16–19 relate more directly to physical processes, as opposed to discretized equations, making them arguably more intuitive. At the control volume interfaces, the values of the dependent variables and their gradients are defined as:

$$c_{k,w} = \alpha_w c_{k,j} + (1-\alpha_w)c_{k,j-1} \quad [20]$$

$$c_{k,e} = \alpha_e c_{k,j+1} + (1-\alpha_e)c_{k,j} \quad [21]$$

$$\nabla c_{k,w} = \beta_w c_{k,j} - \beta_w c_{k,j-1} \quad [22]$$

$$\nabla c_{k,e} = \beta_e c_{k,j+1} - \beta_e c_{k,j} \quad [23]$$

where the coefficients α and β are given by

$$\alpha_w = \frac{\Delta x_{j-1}}{\Delta x_{j-1} + \Delta x_j}; \quad \alpha_e = \frac{\Delta x_j}{\Delta x_j + \Delta x_{j+1}} \quad [24]$$

$$\beta_w = \frac{2}{\Delta x_{j-1} + \Delta x_j}; \quad \beta_e = \frac{2}{\Delta x_j + \Delta x_{j+1}} \quad [25]$$

(Note: If convective transport dominates relative to diffusive transport, then oscillatory numerical behavior can be prevented by using an upwind scheme: $\alpha_w = 0$ and $\alpha_e = 0$ for flow from west to east, and $\alpha_w = 1$ and $\alpha_e = 1$ for flow from east to west.[13]) The coefficients used in BAND, $A^{\circ}_{i,k}$, $B^{\circ}_{i,k}$, $D^{\circ}_{i,k}$ and $G^{\circ}_{i,k}$, can be defined in terms of the quantities defined by Eqs. 16–25.

$$A^{\circ}_{i,k} = (1-\alpha_w)f_{i,k}|_w - \beta_w d_{i,k}|_w \quad [26]$$

$$B^{\circ}_{i,k} = r_{i,k} + \alpha_w f_{i,k}|_w + \beta_w d_{i,k}|_w - (1-\alpha_e)f_{i,k}|_e + \beta_e d_{i,k}|_e \quad [27]$$

$$D^{\circ}_{i,k} = -\alpha_e f_{i,k}|_e - \beta_e d_{i,k}|_e \quad [28]$$

$$G^{\circ}_{i,j} = -(A_{x,w}\mathbf{N}^{\circ}_{i,w} - A_{x,e}\mathbf{N}^{\circ}_{i,e} + \Delta V R_i^{\circ}) \quad [29]$$

It is numerically convenient to set the control volume (ΔV) to zero at the boundaries, which slightly modifies Eqs. 26–29. At the west-side boundary ($j = 1$), there is no "western" interface

$$B^{\circ}_{i,k} = r_{i,k} - (1-\alpha_e)f_{i,k}|_e + \beta_e d_{i,k}|_e \quad [30]$$

$$D^{\circ}_{i,k} = -\alpha_e f_{i,k}|_e - \beta_e d_{i,k}|_e \quad [31]$$

$$G^{\circ}_{i,j} = -(-A_{x,e}\mathbf{N}^{\circ}_{i,e} + \Delta V R_i^{\circ}) \quad [32]$$

and at the east-side boundary ($j = NJ$), there is no "eastern" interface

$$A^{\circ}_{i,k} = (1-\alpha_w)f_{i,k}|_w - \beta_w d_{i,k}|_w \quad [33]$$

$$B^{\circ}_{i,k} = r_{i,k} + \alpha_w f_{i,k}|_w + \beta_w d_{i,k}|_w \quad [34]$$

$$G^{\circ}_{i,j} = -(A_{x,w}\mathbf{N}^{\circ}_{i,w} + \Delta V R^{\circ}_i) \quad [35]$$

J. Deliang Yang showed that writing the governing equations using a physically intuitive control volume approach, i.e. segregating the differential equations into accumulation terms, flux terms, and reaction terms, is compatible with Newman's BAND structure.[9] The remaining difficulty is to efficiently and accurately linearize the governing equations.

Equation linearization.—Certain criteria for a viable algorithmic approach to equation linearization are suggested below:

1. user-input is minimized
2. the process can be applied generally
3. the results are accurate
4. the process is computationally efficient

Methodologies deficient in any of these criteria are programmatically untenable. Several methods for equation linearization are discussed below. Manual, numerical, and symbolic differentiation are each deficient in at least one of the requisite criteria, while automatic differentiation meets these standards for viability.

Manual differentiation has no programming overhead and achieves machine precision, but requires significant user input, is susceptible to human error, and is not generalizable—each unique physical model costs significant personnel time to develop.

Numerical differentiation (or difference quotient) is frequently performed using symmetric difference quotient, see Eq. 36, where g is an arbitrary function of the variables $c_1, \cdots, c_n$. The method requires only a small amount of programming overhead and the process is intuitive. Conceptually, this method can be applied generally, but practically, the optimal step-size, h in Eq. 36, is difficult to determine in a general case.[18–21] Choosing the optimal step-size can be costly in terms of personnel time or require sophisticated programming, which can decrease computational performance. Non-optimal values of h lead to inaccuracies in the form of systematic precision or truncation errors. Newman introduced AUTOBAND to perform numerical differentiation, but noted its vulnerability to inaccuracies and loss in computational performance.[8] Even an optimally chosen value of h, only accurately computes two-thirds of the significant digits of $\partial g/\partial c_i$.[22]

$$\frac{\partial g}{\partial c_i} \approx \frac{g(c_1, \cdots, c_i + h, \cdots, c_n) - g(c_1, \cdots, c_i - h, \cdots, c_n)}{2h} \quad [36]$$

Symbolic differentiation is attractive because user-friendly software packages such as SymPy and Mathematica already exist to perform this type of differentiation, therefore there is little programming overhead for the general user. However, symbolic differentiation is prone to producing overly complicated representations of the derivative, called expression swell,[23] which can lead to costly evaluations of the derivative. In addition, it can be difficult to produce a symbolic derivative of a function that cannot be expressed in closed form.[24]

While there are a number of different automatic differentiation approaches,[24,25] this paper will focus on **dual number automatic differentiation**. This process uses a bottom-up approach to differentiation: though there are an infinite number of possible functions that can be programmed, there are only a finite number of elementary functions (sin, cos, log, exp, etc.) and elementary operations (addition, subtraction multiplication, division, etc.) within a programming language. If the derivatives of these elementary operators and functions can be programmed, then by applying the chain-rule, the derivative of any arbitrary function can be calculated because it is composed of these elementary functions and operations. The next Section illustrates how dual numbers are used to conserve the chain-rule. The major drawback of this method is that it requires extensive programming overhead: the definition of a new numeric structure (`type(dual)`) and definitions for how elementary operators, elementary functions, and existing data types interact with this new numeric structure; a process frequently referred to as operator overloading. Fortunately, the extensive work of operator overloading has already been accomplished for a variety of programming languages[4,26–29] and this work has been compiled into respective libraries and modules. For the general scientific user, this means that automatic differentiation is achievable by simply importing a library or copying a module.

Dual number automatic differentiation.—A comprehensive analysis of dual numbers and their applications can be found in the literature.[30–32] And while a rigorous mathematical description of these concepts is beyond the scope of this paper, a brief overview of the properties of dual numbers serves to illustrate their utility in the context of automatic differentiation. A dual number u_i can be represented as

$$u_i = c_i + \dot{c}_i\epsilon \quad [37]$$

where c_1 and $\dot{c}_1$ are real numbers, and ϵ is a nilpotent number (i.e. a small non-zero number such that, $\epsilon^2 = \epsilon^3 = \cdots = 0$). Addition and multiplication of dual numbers are defined by Eqs. 38 and 39, respectively

$$\begin{aligned} u_1 + u_2 &= (c_1 + \dot{c}_1\epsilon) + (c_2 + \dot{c}_2\epsilon) \\ &= (c_1 + c_2) + (\dot{c}_1 + \dot{c}_2)\epsilon \end{aligned} \quad [38]$$

$$\begin{aligned} u_1 \cdot u_2 &= (c_1 + \dot{c}_1\epsilon)(c_2 + \dot{c}_2\epsilon) \\ &= c_1c_2 + (c_2\dot{c}_1 + c_1\dot{c}_2)\epsilon \end{aligned} \quad [39]$$

where $\dot{c}_1\dot{c}_2\epsilon^2 = 0$ because $\epsilon^2 = 0$.

Using a Taylor series expansion, one can evaluate a function with a dual number as input:

$$\begin{aligned} f(c_1 + \dot{c}_1\epsilon) &= f(c_1) + f'(c_1)\dot{c}_1\epsilon \\ &\quad + \frac{1}{2}f''(c_1)\dot{c}_1^2\epsilon^2 + \ldots\epsilon^3 + \ldots \\ &= f(c_1) + f'(c_1)\dot{c}_1\epsilon \end{aligned} \quad [40]$$

The chain rule is also preserved through the process outlined in Eq. 40:

$$\begin{aligned} f[g(c_1 + \dot{c}_1\epsilon)] &= f[g(c_1) + g'(c_1)\dot{c}_1\epsilon] \\ &= f[g(c_1)] + f'[g(c_1)]g'(c_1)\dot{c}_1\epsilon \end{aligned} \quad [41]$$

The conservation of the chain rule shown in Eq. 41 illustrates that dual numbers can be applied generally to evaluate numerical derivatives. By setting $\dot{c}_i = 1$ one can simultaneously evaluate both the function, $f(c_i)$, and its derivative, $f'(c_i)$. Conceptually, this means that by programming a function one is also implicitly defining its derivative—a useful simplification for numerical programming.

As can be seen from Eqs. 10–12, the partial derivatives of the flux, generation, and accumulation expressions are necessary inputs to the BAND subroutine. However, through the use of dual numbers, the user is completely absolved from linearizing the system of equations; by programming the analytical expressions for flux, generation, and accumulation, the linearization of the equations is accomplished implicitly through dual number automatic differentiation.

Merging Automatic Differentiation with BAND

In the Sections above, an overview of BAND, the control volume formulation, and dual number automatic differentiation were given,

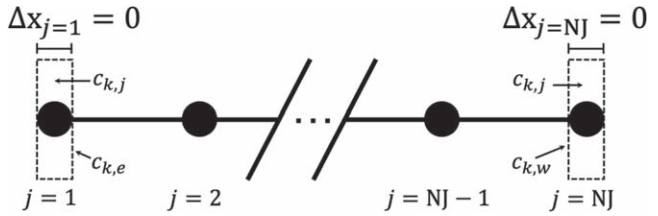

Figure 2. Diagram of numerical approach used at the boundaries.

with the intention of giving the reader a high-level understanding of how these techniques can be combined to simplify the process of numerical programming. The following Sections provide a detailed explanation of how to implement dual number automatic differentiation. When discussing these dual expressions, braces have been used to highlight connections between the dual expressions and Eqs. 16–19. In addition, large Sections of the code itself are discussed and an example problem using porous electrode theory is used to guide the reader through the implementation process.

Representing the dependent variables using dual numbers.—As can be seen from Eqs. 11–14, the finite volume formulation produces two types of change variables: one with respect to concentration, Δc_k, and a second with respect to the spatial concentration gradient, $\Delta(\nabla c_k)$. The total number of change variables then is $2N$, where N is the number of dependent variables: N terms of c_k and N terms of ∇c_k. The dual variables of c_k and ∇c_k can be constructed from their real counterparts, and structured into arrays:

$$c_k = \left(c_k^{real}, \overbrace{\left(\frac{\partial c_k}{\partial c_1}, \cdots, \frac{\partial c_k}{\partial c_N}, \ \frac{\partial c_k}{\partial(\nabla c_1)}, \cdots, \frac{\partial c_k}{\partial(\nabla c_N)} \right)}^{dx} \right] \quad [42]$$

$$\nabla c_k = \left(\nabla c_k{}^{real}, \left(\frac{\partial(\nabla c_k)}{\partial c_1}, \cdots, \frac{\partial(\nabla c_k)}{\partial c_N}, \ \frac{\partial(\nabla c_k)}{\partial(\nabla c_1)}, \cdots, \frac{\partial(\nabla c_k)}{\partial(\nabla c_N)} \right) \right) \quad [43]$$

The terms inside the inner parentheses form the dx array, where

$$\frac{\partial c_k}{\partial(\nabla c_j)} = \frac{\partial(\nabla c_k)}{\partial c_j} = 0 \quad [44]$$

and

$$\frac{\partial c_k}{\partial c_j} = \frac{\partial(\nabla c_k)}{\partial(\nabla c_j)} = \delta_{kj} \quad [45]$$

where δ_{kj} is the Kronecker delta function; i.e. dx is a sparse array composed entirely of zeros except for a single value of 1.

Representing the governing equations using dual numbers.—The dual output of a function of these change variables has a similar structure; for example the flux, $\mathbf{N}_i$, takes the form:

$$\mathbf{N}_i(c_1,\cdots,c_N, \nabla c_1,\cdots,\nabla c_N) = \left(\mathbf{N}_i^\circ, \left(\underbrace{\frac{\partial \mathbf{N}_i}{\partial c_1}, \cdots, \frac{\partial \mathbf{N}_i}{\partial c_N}}_{f_{i,k}}, \ \underbrace{\frac{\partial \mathbf{N}_i}{\partial(\nabla c_1)}, \cdots, \frac{\partial \mathbf{N}_i}{\partial(\nabla c_N)}}_{d_{i,k}} \right) \right) \quad [46]$$

where $\mathbf{N}_i^\circ$ is the value of the flux evaluated at the state $(c_1,\cdots,c_N, \nabla c_1,\cdots,\nabla c_N)$, and the dx array is composed of the partial derivatives of the flux with respect to the change variables and has a length of $2N$. The dx array can be further segregated into partial derivatives of the form $\partial \mathbf{N}_i/\partial c_k$, and $\partial \mathbf{N}_i/\partial(\nabla c_k)$, which correspond to the coefficients $f_{i,k}$ and $d_{i,k}$ from Eqs. 16 and 17.

The generation, R_i, and accumulation, Accum_i, expressions are not functions of spatial concentration gradients and therefore the trailing zeros in Eqs. 47 and 50 correspond to the partial derivatives $\partial R_i/\partial(\nabla c_k)$ and $\partial(\text{Accum}_i)/\partial(\nabla c_k)$, respectively.

$$R_i(c_1,\cdots,c_N) = \left(R_i^\circ, \left(\underbrace{\frac{\partial R_i}{\partial c_1}, \cdots, \frac{\partial R_i}{\partial c_N}}_{r_{i,k}^{\{rxn\}}}, \ 0, \cdots, 0 \right) \right) \quad [47]$$

The accumulation term, Accum_i, takes the form

$$\text{Accum}_i = \frac{\partial(h_i)}{\partial t} \quad [48]$$

where h_i is an arbitrary function of the dependent variables, c_k

$$h_i(c_1,\cdots,c_N) = \left(h_i^\circ, \left(\frac{\partial h_i}{\partial c_1}, \cdots, \frac{\partial h_i}{\partial c_N}, \ 0, \cdots, 0 \right) \right) \quad [49]$$

Noting that

$$\frac{\partial h_i^\circ}{\partial t} = 0$$

and using the numerical approximation

$$\frac{\partial c_k}{\partial t} \approx \frac{\Delta c_k}{\Delta t}$$

Accum_i can be represented as

$$\text{Accum}_i = \left(0, \left(\underbrace{\frac{1}{\Delta t}\frac{\partial h_i}{\partial c_1}, \cdots, \frac{1}{\Delta t}\frac{\partial h_i}{\partial c_N}}_{r_{i,k}^{\{acc\}}}, \ 0, \cdots, 0 \right) \right) \quad [50]$$

which allows one to evaluate $r_{i,k}$ (Eq. 18) in terms of the generation and accumulation terms.

$$r_{i,k} = (r_{i,k}^{\{rxn\}} - r_{i,k}^{\{acc\}})\Delta V \quad [51]$$

As a reminder the real portions of the governing expressions, N_i° and R_i° are used in the expression for $g_{i,j}^\circ$ (Eq. 19). The reader may notice in Eq. 50 that the value h_i° - i.e. the real portion of h_i - may simply be ignored because it is not utilized in the control volume formulation. However, in representing boundary conditions, there is utility in explicitly defining $h_i^\circ = 0$.

Representing boundary conditions using dual numbers.—As stated previously, and as can be seen in Fig. 2, at the boundaries the control volume is set to zero for numerical convenience. The effect of setting $\Delta x = 0$ (i.e. $\Delta V = 0$) can be inferred from Eqs. 20, 21, and 24: at $j = 1$, $c_{k,1} = c_{k,e}$ and at j=NJ, $c_{k,NJ} = c_{k,w}$. This provides another convenience for the user because one does not need to distinguish between control volume variables and interface variables at the boundaries; this allows the boundary conditions to be defined in terms of the interface variables, which is compatible for a wide range of physical conditions.

The boundary conditions commonly take one of three forms: 1) a specified flux expression, 2) a specified concentration expression, or

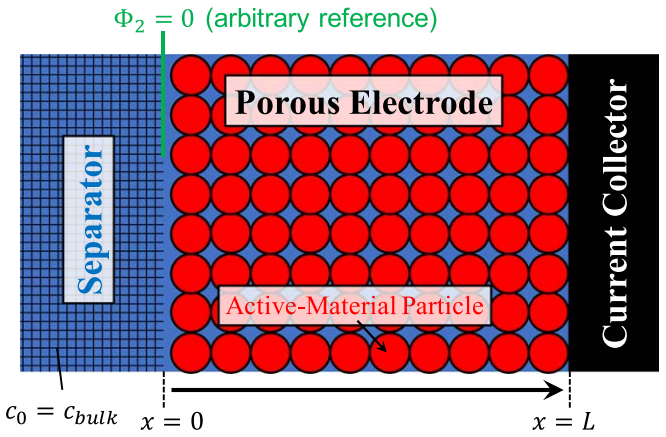

Figure 3. Schematic of the simple porous-electrode system used to demonstrate the utility of combining BAND with a automatic differentiation. At $x = 0$, the porous-electrode is in contact with the separator, where the concentration of Li^+ in the electrolyte is assumed to remain constant at its bulk concentration, c_{bulk}; the electrochemical potential in the solution is set to an arbitrary reference potential of 0, $\Phi_2 = 0$. At the other end, $x = L$, the electrode is in contact with an electrically conductive current collector. The equations used to model this system are outlined in Table I and the parametric information is provided in Table II.

3) in the case that there are no spatial gradients in the governing equation, the governing equation is repeated at the boundary:

$$\mathbf{N} = p \quad [52]$$

$$c = q \quad [53]$$

$$\frac{\partial h}{\partial t} = s \quad [54]$$

where p, q, and s are arbitrary functions of time and the dependent variables. To make these expressions algorithmically compatible with dual number representations, they need to be written as homogeneous expressions:

$$BC_1 = \mathbf{N} - p \quad [55]$$

$$BC_2 = c - q \quad [56]$$

$$BC_3 = \frac{\partial h}{\partial t} - s \quad [57]$$

The dual representation of the boundary conditions, BC_i, can then be readily decomposed into the relevant matrix coefficients.

Table I. A summary of the mathematical expressions used to simulate the performance of a hypothetical porous cathode material.

Separator	Governing equations	Current Collector
$c_+ = c_{bulk}$	$\epsilon\frac{\partial c_+}{\partial t} = -\epsilon^{1.5}\nabla\cdot\mathbf{N}_+ + \frac{ai_n}{F}$	$\mathbf{N}_+ = 0$
—	$(1-\epsilon)\frac{\partial c_x}{\partial t} = \frac{-ai_n}{F}$	—
$\mathbf{i}_1 = 0$	$0 = (1-\epsilon)\sigma\nabla^2\Phi_1 - ai_n$	$\mathbf{i}_1 = -\mathbf{i}_{app}$
$\Phi_2 = 0$	$0 = -\epsilon^{1.5}F\sum_i z_i\nabla\cdot\mathbf{N}_i + ai_n$	$\mathbf{i}_2 = 0$
	Thermodynamics	
	$U = U_0 + \frac{RT}{F}\ln\left[\left(\frac{c_0}{c_{bulk}}\right)\left(\frac{(1-\theta)}{\theta}\right)\right]$	$\theta = \frac{c_x}{c_{x,\max}}$
	Electrochemical Reaction Kinetics	
	$i_n = i_0\left[\exp\left(\frac{\alpha_a F\eta}{RT}\right) - \exp\left(\frac{-\alpha_c F\eta}{RT}\right)\right]$	$\eta = \Phi_1 - \Phi_2 - U$
	$i_0 = Fk_{rxn}c_0^{\alpha_a}c_x^{\alpha_c}(c_{x,\max} - c_x)^{\alpha_a}$	

Table II. Physical parameters used to simulate the electrode performance illustrated in Fig. 11.

Solid-State Properties		
Electrode thickness	$L_{elect} = 100$	m
Electronic conductivity	$\sigma = 3 \times 10^{-4}$	S cm^{-1}
Porosity	$\epsilon = 0.4$	—
Active-material size (radius)	$R_x = 500$	nm
Electrochemical surface area	$a = 3(1-\epsilon)/R_x$	cm^2 cm^{-3}
Material density	$\rho = 5$	g cm^{-3}
Material specific capacity	$Q_{x,\max} = 200$	mAh g^{-1}
Solution Properties		
Electrolyte concentration	$c_{bulk} = 1 \times 10^{-3}$	mol cm^{-3}
Ionic Diffusion coefficients	$D_{Li^+} = D_{PF_6^-} = 1 \times 10^{-7}$	cm^2 s^{-1}
Species charge	$z_{Li^+} = -z_{PF_6^-} = +1$	—
Electrochemical Reaction Parameters		
Standard voltage (vs. Li)	$U_0 = 3.3$	V
Charge-transfer coefficients	$\alpha_a = \alpha_c = 0.5$	—
Electrochemical reaction rate constant	$k_{rxn} = 1 \times 10^{-8}$	$cm^{5/2}$ $mol^{-1/2}$ s^{-1}
Maximum solid-state concentration	$c_{x,\max} = \rho Q_{x,\max}/F$	mol cm^{-3}
Temperature	$T = 298$	K
Ideal Gas Constant	R=8.314	J mol^{-1} K^{-1}
Farraday Constant	$F = 96485$	C mol^{-1}

```
program unsteady
  use user_input
  use variables, only : cprev, delC
  use write_data_mod
  use echem_mod
  use experiment_mod

  implicit none
  integer :: it
  integer :: i, k, j

  call initial_condition()

  do it = 1, Numbertimesteps

    do j = 1, NJ
      call auto_fill(j)
      call ABDGXY(j)
      call BAND(j)
    end do

    cprev = cprev + delC              ! Update the dependent variables
    time  = time  + delT              ! Update the time

  end do

end program unsteady
```

Figure 4. Program `unsteady` uses `auto_fill`, `ABDGXY`, and `BAND` to iteratively solve an unsteady (time varying) system of equations.

```
! ------------------------------------------------------------------------
! functions to convert concentration and dcdx variables from REAL to DUAL
! ------------------------------------------------------------------------
function c_to_dual(c_vars)          result (c_dual)
  type(dual), dimension(N)       :: c_dual
  real, dimension(N), intent(in) :: c_vars
  real, dimension(N*2)           :: dx_array
  integer :: ic

  do ic = 1, N
    dx_array = 0.0              ! set the dx_array to zero (all elements)
    dx_array(ic) = 1.0

    c_dual(ic) = DUAL(c_vars(ic), dx_array)
  end do
end function c_to_dual


function dcdx_to_dual(dcdx)         result (dcdx_dual)
  type(dual), dimension(N)       :: dcdx_dual
  real, dimension(N), intent(in) :: dcdx
  real, dimension(N*2)           :: dx_array
  integer :: ic

  do ic = 1, N
    dx_array = 0.0              ! set the dx_array to zero (all elements)
    dx_array(N+ic) = 1.0

    dcdx_dual(ic) = DUAL(dcdx(ic), dx_array)
  end do
end function dcdx_to_dual
```

Figure 5. The functions `c_to_dual` and `dcdx_to_dual` convert c_k and ∇c_k to dual numbers.

$$BC_i(c_1,\cdots,c_k,\nabla c_1,\cdots,\nabla c_k) = \left(\underbrace{BC_i^\circ}_{g_{i,j}^\circ}, \left(\underbrace{\frac{\partial BC_i}{\partial c_i},\cdots,\frac{\partial BC_i}{\partial c_k}}_{f_{i,k}}, \underbrace{\frac{\partial BC_i}{\partial(\nabla c_i)},\cdots,\frac{\partial BC_i}{\partial(\nabla c_k)}}_{d_{i,k}}\right)\right) \quad [58]$$

BC_3 represents the specific case where setting $h_i^\circ = 0$ is of practical utility. By inspection of Eq. 54, it can be seen that $g_{3,j}^\circ = -s$; this is achieved if and only if $h_i^\circ = 0$. Simply using Eq. 49 would erroneously produce $g_{3,j}^\circ = h^\circ/\Delta t - s$, which is not mathematically or physically consistent with Eq. 54.

Example problem.—The following brief example may be useful to the reader in understanding the general expressions outlined above. The problem explored here is a porous-electrode with solid-solution active-material particles soaked in a dilute electrolyte composed of a monovalent binary salt completely dissolved and dissociated in a solvent. One end of the electrode is in contact with a separator, and the other end is attached to an electronic current collector (Fig. 3). Two main assumptions are used to simplify the problem. First, the solid-state radial diffusion within the insertion particles is assumed very fast and thus transport resistances within the particles are neglected. Second, the electrolyte concentration at the electrode/separator interface is assumed time invariant; polarization induced by the transport phenomena in the separator and the kinetics of the counter electrode are not considered. It is assumed that there are variations only along the length of the electrode (from the separator to the current collector). The model as described involves 4 dependent variables: the concentration of Li^+ in the solution (c_0), the concentration of Li in the solid-state (c_x), the electric potential in the solid-state (Φ_1), and the electrochemical potential in the solution, (Φ_2). Simulations are run for the continuous constant-current insertion of Li^+ into the solid-state starting from a fully delithiated state. The instantaneous electrode potential at the current-collector interface, i.e. $\Phi_1|_{x=L}$, is monitored and used to control the end of the insertion process.

```
function FLUX(c_vars_dual, dcdx_vars_dual) result(Flux_)
  ! (1)   N_0 = -eps**1.5 * (D_Li * dc0/dx + z_Li * u_Li * c0 * F * dPhi_2/dx)
  ! (2)   N_x = 0
  ! (3)   i_1 = -(1-eps) * sigma * dPhi_1/dx
  ! (4)   i_2/F = -eps**1.5 * [ (z_1 * diff_1 + z_2 * diff_2)*dc0/dx
  !                      (z_1**2 * u_1 + z_2**2 * u_2) * F * c0 * dPhi_2/dx]
  !       i.e. i_2 / F = sum_i (z_i * N_i)
  type(dual), dimension(N)              :: Flux_
  type(dual), dimension(N), intent(in)  :: c_vars_dual, dcdx_vars_dual
  type(dual)                            :: diff_
  real                                  :: u_1, u_2    ! mobility

  c0    = c_vars_dual(1)
  c_x   = c_vars_dual(2)
  Phi_1 = c_vars_dual(3)
  Phi_2 = c_vars_dual(4)

  dc0dx    = dcdx_vars_dual(1)
  dc_xdx   = dcdx_vars_dual(2)
  dPhi_1dx = dcdx_vars_dual(3)
  dPhi_2dx = dcdx_vars_dual(4)

  ! Nernst-Einstein Relationship
  u_1 = diff_Li / (Rigc * Temp)
  u_2 = diff_PF6 / (Rigc * Temp)

  Flux_(1) = -porosity**1.5 * (diff_Li * dc0dx + z_1 * u_1 * c0 * Fconst * dPhi_2dx)
  Flux_(2) = 0.0
  Flux_(3) = -(1 - porosity) * sigma * dPhi_1dx
  Flux_(4) = -porosity**1.5 * Fconst * ( (z_1 * diff_Li + z_2 * diff_PF6) * dc0dx &
      + (z_1**2 * u_1 + z_2**2 * u_2) * Fconst * c0 * dPhi_2dx )

end function FLUX

! ********************************************************************************
function RXN(c_vars_dual) result(Rxn_)
  type(dual), dimension(N)               :: Rxn_
  type(dual), dimension(N), intent(in)   :: c_vars_dual
  type(dual) :: i_rxn

  c0    = c_vars_dual(1)
  c_x   = c_vars_dual(2)
  Phi_1 = c_vars_dual(3)
  Phi_2 = c_vars_dual(4)

  i_rxn = Butler_Volmer_Rxn(c0, c_x, Phi_1, Phi_2)

  Rxn_(1) = +volumetric_surface_area * i_rxn / Fconst     !  a * i_rxn / F
  Rxn_(2) = -volumetric_surface_area * i_rxn / Fconst     ! -a * i_rxn / F
  Rxn_(3) = -volumetric_surface_area * i_rxn              ! -a * i_rxn
  Rxn_(4) = +volumetric_surface_area * i_rxn              !  a * i_rxn

end function RXN

! ********************************************************************************
function ACCUM(c_vars_dual) result(Accum_)
  type(dual), dimension(N)              :: Accum_
  type(dual), dimension(N), intent(in)  :: c_vars_dual

  c0    = c_vars_dual(1)
  c_x   = c_vars_dual(2)
  Phi_1 = c_vars_dual(3)
  Phi_2 = c_vars_dual(4)

  Accum_(1) = (porosity) * c0/delT               ! eps * dc0/dt
  Accum_(2) = (1.0 - porosity) * c_x/delT        ! (1-eps) * dc_x/dt
  Accum_(3) = 0.0                ! (electroneutrality - no accumulation of e-)
  Accum_(4) = 0.0                ! (electroneutrality - no accumulation of ions)

  Accum_%x  = 0.0                ! set real part of Accum to 0

end function ACCUM
```

Figure 6. Programmatic representation of the finite volume method (FVM) or control volume approach: flux (`FLUX`), generation (`RXN`), and accumulation (`ACCUM`). The real portion of the accumulation term is explicitly set to zero (`Accum_%x = 0.0`) for programmatic simplicity; there is no physical interpretation for this term. These expressions are used in `auto_fill` to calculate the matrix coefficients $d_{i,k}$, $f_{i,k}$, $r_{i,k}$, and $g^{\circ}_{i,j}$ - see Fig. 9 lines 77–101.

Mathematical expressions.—Assuming dilute solution theory and no convection, the mass flux of specie i is driven by Fickian diffusion and ionic migration

$$\mathbf{N}_i = -D_i \nabla c_i - z_i u_i F c_i \nabla \Phi_2 \quad [59]$$

where the specie mobility, u_i is assumed to follow the Nernst-Einstein relation:

$$u_i = \frac{D_i}{RT} \quad [60]$$

The current in the solid-state, $\mathbf{i}_1$, follows Ohm's law

$$\mathbf{i}_1 = -\sigma \nabla \Phi_1 \quad [61]$$

and the solution (electrolytic) current, $\mathbf{i}_2$, is facilitated by the flux of ions

$$\mathbf{i}_2 = F \sum_i z_i \mathbf{N}_i \quad [62]$$

For this problem it is assumed that the porosity, ϵ, of the electrode does not vary spatially or temporally, and the tortuosity, τ,

```
function Boundary_WEST (c_vars_dual, dcdx_vars_dual) result(BC_WEST_)
  ! Separator Interface
  type(dual), dimension(N)               :: BC_WEST_
  type(dual), dimension(N), intent(in)   :: c_vars_dual, dcdx_vars_dual

  c0    = c_vars_dual(1)
  c_x   = c_vars_dual(2)
  Phi_1 = c_vars_dual(3)
  Phi_2 = c_vars_dual(4)

  flux_temp   = FLUX(c_vars_dual, dcdx_vars_dual)
  accum_temp  = ACCUM(c_vars_dual)
  rxn_temp    = RXN(c_vars_dual)

  BC_WEST_(1) = c0 - cbulk                        ! c_0 = cbulk
  BC_WEST_(2) = accum_temp(2) - rxn_temp(2)       ! (1-eps) dc/dt = -a * i_rxn/F
  BC_WEST_(3) = flux_temp(3) - 0.0                ! i_1 = 0
  BC_WEST_(4) = Phi_2 - 0.0                       ! Phi_2 = 0

end function Boundary_WEST

function Boundary_EAST (c_vars_dual, dcdx_vars_dual) result(BC_EAST_)
  ! Current Collector Interface
  type(dual), dimension(N)               :: BC_EAST_
  type(dual), dimension(N), intent(in)   :: c_vars_dual, dcdx_vars_dual

  c0    = c_vars_dual(1)
  c_x   = c_vars_dual(2)
  Phi_1 = c_vars_dual(3)
  Phi_2 = c_vars_dual(4)

  flux_temp   = FLUX(c_vars_dual, dcdx_vars_dual)
  accum_temp  = ACCUM(c_vars_dual)
  rxn_temp    = RXN(c_vars_dual)

  BC_EAST_(1) = flux_temp(1) - 0.0                ! N_+ = 0
  BC_EAST_(2) = accum_temp(2) - rxn_temp(2)       ! (1-eps) dc/dt = -a * i_rxn/F
  BC_EAST_(3) = flux_temp(3) - applied_current_A  ! i_1 = i_applied
  BC_EAST_(4) = flux_temp(4) - 0.0                ! i_2 = 0

end function Boundary_EAST
```

Figure 7. Programmatic representation of the western (`Boundary_WEST`) and eastern (`Boundary_EAST`) boundary conditions. Look at `function Boundary_WEST`, three commonly utilized types of boundary conditions are shown: (1) specified concentration, (2) repetition of the governing equation, (3) specified flux.

is assumed to follow the Bruggeman relationship:

$$\tau = \epsilon^{-0.5} \quad [63]$$

The effective flux of within the electrolyte is proportional to the porosity and inversely proportional to the tortuosity:

$$\mathbf{N}_{i,eff} = \frac{\epsilon}{\tau}\mathbf{N}_i \quad [64]$$

The insertion reaction at the particle/electrolyte interface is assumed to follow Butler-Volmer kinetics, while the electrochemical thermodynamics are assumed to follow a Nernstian relationship. Detailed expressions of the governing equations, boundary conditions, electrochemical reaction kinetics and thermodynamics can be found in Table I, while the parametric information is provided in Table II.

Programmatic representations.—The necessary equations to integrate dual number automatic differentiation with `BAND` have been reported. To streamline the adoption of this process functional Fortran subroutines and functions are provided. The details of the provided code solve the equations outlined in Table I. The subroutines `BAND` and `MATINV` are freely available online [33] as well as in Appendix C of *Electrochemical Systems* 3rd Ed.[8] `dnadmod` is also available online Ref. 34 and so these Sections of code will not be repeated here.

The main iterative loop to solve a system of partial differential equations is given in Fig. 4. The program `unsteady` uses the subroutines `auto_fill`, `ABDGXY`, and `BAND` to solve for `delC` and then update the values of the dependent variables, `cprev`.

The subroutine `auto_fill` takes $c_{k,j}$ as an input, and returns the values α_w, α_e, β_w, β_e, $d_{i,k}|_w$, $d_{i,k}|_e$, $f_{i,k}|_w$, $f_{i,k}|_e$, $r_{i,k}$, and $g^{\circ}_{i,j}$. The sequence of calculations `auto_fill` performs are provide below:

1. calculate α_w, α_e, β_w, β_e
2. calculate $c_{k,w}$, $c_{k,e}$, $\nabla c_{k,w}$, $\nabla c_{k,e}$
3. convert $c_{k,j}$, $c_{k,w}$, $c_{k,e}$, $\nabla c_{k,w}$, $\nabla c_{k,e}$ to dual numbers
4. Evaluate the governing equations: $\mathbf{N}_i$, R_i, Accum$_i$, or appropriate boundary conditions
5. Calculate $d_{i,k}|_w$, $d_{i,k}|_e$, $f_{i,k}|_w$, $f_{i,k}|_e$, $r_{i,k}$, $g^{\circ}_{i,j}$ from the governing equations and boundary conditions

Steps 1 and 2 follow Eqs. 24, 25, and 20–23 (see lines 33-37 and 44-48 of `auto_fill`). Step 3 is carried out through the functions `c_to_dual` and `dcdx_to_dual` given in Fig. 5; these two functions correspond to Eqs. 42 and 43, respectively, and the values of `dx_array` (dx) follow from Eqs. 44 and 45.

The governing equations are evaluated using the functions `FLUX`, `RXN`, and `ACCUM` given in Fig. 6, and the boundary conditions are evaluated using `Boundary_WEST` and `Boundary_EAST` given in Fig. 7. The dual number results of these functions are used to calculate the matrix coefficients in step 5.

Step 5 is performed in lines 59–101 of `auto_fill` using conditional statements to differentiate between the western and eastern boundary conditions, as well as the governing equations; these calculations are done in accordance with Eqs. 16–19.

The governing equations need to be segregated into their flux, generation, and accumulation components as outlined by Eqs. 9 and 10 and as can be seen from Fig. 6, their programmatic representations easily relate to their mathematical forms. It is important to note that for the functions `FLUX`, `RXN`, and `ACCUM`, the input variables as well as the resulting output need to be defined as `type(dual)`, with dimension, N. In addition, any intermediate calculations that are functions of the change variables need to be defined as `type (dual)` as well—see the electrochemical reaction rate, `i_rxn` in `function RXN`.

In instances where the physics are complex it can be advantageous to use supplemental modules to contain the equations that

```
module echem_mod
  use user_input, only: Rigc, Fconst, Temp, max_Li_conc, cbulk
  use dnadmod

  implicit none

  real :: alpha_a = 0.5
  real :: alpha_c = 0.5
  real :: k_rxn   = 1.0e-8

contains

! ================================================================================
  function OCP(c0, c_x) result(OpenCircuitVoltage)
    type(dual)                :: OpenCircuitVoltage
    type(dual), intent(in)    :: c0, c_x
    type(dual)                :: theta
    real                      :: U_ref

    theta = c_x / max_Li_conc

    U_ref =  3.3

    OpenCircuitVoltage = U_ref + Rigc*Temp/Fconst * LOG( c0/cbulk * (1.0 - theta)/theta )

  end function OCP

! ================================================================================
  function exchange_current_density(c0, c_x)  result(ex_curr)
    type(dual)              :: ex_curr
    type(dual), intent(in)  :: c0, c_x

    ex_curr = Fconst * k_rxn * c0**alpha_a * c_x**alpha_c * (max_Li_conc - c_x)**alpha_a

  end function exchange_current_density

! ================================================================================
  function Butler_Volmer_Rxn(c0, c_x, Phi_1, Phi_2)    result(i_BV)

    type(dual)              :: i_BV
    type(dual), intent(in)  :: c0, c_x, Phi_1, Phi_2
    type(dual)              :: eta, U_ocp
    type(dual)              :: i_ex_curr

    i_ex_curr = exchange_current_density(c0, c_x)

    U_ocp = OCP(c0, c_x)
    eta   = Phi_1 - Phi_2 - U_ocp

    i_BV = i_ex_curr * ( EXP(alpha_a * Fconst * eta / (Rigc * Temp)) &
                       - EXP(-alpha_c * Fconst * eta / (Rigc * Temp)) )

  end function Butler_Volmer_Rxn

end module echem_mod
```

Figure 8. Programmatic representation of the electrochemical reaction expressions, given in Table I. Using dual numbers to represent these expressions eliminates the need to derive and program functions to evaluate the partial derivatives of these expressions, i.e. $\partial U/\partial c_0$, $\partial U/\partial c_x$, etc.

```
subroutine auto_fill(j)
  use dnadmod
  use user_input, only : N, NJ
  use GOV_EQNS
  use variables, only: cprev, delX

  implicit none
  integer, intent(in) :: j   ! node (mesh point) index

  integer :: ic           ! loop indices

  ! variables and their derivatives at the control volume interfaces
  real, dimension(N) :: cW, cE, dcdxW, dcdxE

  ! dual variables
  type(dual), dimension(N) :: cW_dual, dcdxW_dual, flux_dualW
  type(dual), dimension(N) :: cE_dual, dcdxE_dual, flux_dualE
  type(dual), dimension(N) :: cj_dual, reaction_dual, accumulation_dual
  type(dual), dimension(N) :: boundary_conditionW, boundary_conditionE

  ! set all matrix variables (dW, dE, fW, fE, rj, smG) to 0.0
  dW = 0.0
  dE = 0.0
  fW = 0.0
  fE = 0.0
  rj = 0.0
  smG = 0.0

  !--------------------------------------------------------------------------
  ! Calculate cW, cE, dcdxW, dcdxE
  !--------------------------------------------------------------------------
  if (j /= 1) then                          !---------------------------------
    alphaW = delx(j-1)/(delx(j-1)+delx(j))  ! West Side Interface Variables:
    betaW = 2.0/(delx(j-1)+delx(j))         ! cW, dcdxW
    do ic=1,N                               !---------------------------------
      cW(ic)    = alphaW*cprev(ic,j) + (1.0 - alphaW)*cprev(ic,j-1)
      dcdxW(ic) = betaW * (cprev(ic,j) - cprev(ic,j-1))
    end do
    cW_dual = c_to_dual(cW)
    dcdxW_dual = dcdx_to_dual(dcdxW)
  end if
                                            !---------------------------------
  if (j /= NJ) then                         ! East Side Interface Variables:
    alphaE = delx(j)/(delx(j+1)+delx(j))    ! cE, dcdxE
    betaE = 2.0/(delx(j)+delx(j+1))         !---------------------------------
    do ic=1,N
      cE(ic)    = alphaE*cprev(ic,j+1) + (1.0 - alphaE)*cprev(ic,j)
      dcdxE(ic) = betaE * (cprev(ic,j+1) - cprev(ic,j))
    end do
    cE_dual = c_to_dual(cE)
    dcdxE_dual = dcdx_to_dual(dcdxE)
  end if


  !--------------------------------------------------------------------------
  ! Boundary Conditions
  !--------------------------------------------------------------------------
  if (j == 1) then                          ! West Side Boundary Conditions
    boundary_conditionW = Boundary_WEST(cE_dual, dcdxE_dual) !---------------
    do ic = 1,N
        fE(ic, :) = boundary_conditionW(ic)%dx(1:N)
        dE(ic, :) = boundary_conditionW(ic)%dx(N+1:2*N)

        smG(ic)   = -(-boundary_conditionW(ic)%x)
    end do
                                            !-----------------------------
  else if (j == NJ) then                    ! East Side Boundary Conditions
      boundary_conditionE = Boundary_EAST(cW_dual, dcdxW_dual) !-------------
    do ic = 1,N
        fW(ic, :) = boundary_conditionE(ic)%dx(1:N)
        dW(ic, :) = boundary_conditionE(ic)%dx(N+1:2*N)

        smG(ic)   = -(boundary_conditionE(ic)%x)
    end do

  !--------------------------------------------------------------------------
  ! Governing Equations
  !--------------------------------------------------------------------------
  else                                      !------------ Fluxes -------------
    flux_dualW = FLUX(cW_dual, dcdxW_dual)  ! West Side Flux
    flux_dualE = FLUX(cE_dual, dcdxE_dual)  ! East Side Flux

    cj_dual = c_to_dual(cprev(:,j))         !-------------------------------
    reaction_dual = RXN(cj_dual)            ! Control Volume - Rxn, Accum
    accumulation_dual = ACCUM(cj_dual)      !-------------------------------

    do ic = 1,N                             ! ic - equation number
        fW(ic,:) = flux_dualW(ic)%dx(1:N)     * Crx_Area(1,j)
        fE(ic,:) = flux_dualE(ic)%dx(1:N)     * Crx_Area(2,j)
        dW(ic,:) = flux_dualW(ic)%dx(N+1:2*N) * Crx_Area(1,j)
        dE(ic,:) = flux_dualE(ic)%dx(N+1:2*N) * Crx_Area(2,j)

        rj(ic,:) = ( reaction_dual(ic)%dx(1:N)                &
                 &  - accumulation_dual(ic)%dx(1:N) )*Cntrl_Vol(j)

        smG(ic)  = -(   flux_dualW(ic)%x * Crx_Area(1,j)      &
                 &    - flux_dualE(ic)%x * Crx_Area(2,j)      &
                 &    + reaction_dual(ic)%x * Cntrl_Vol(j) )
    end do
  end if

end subroutine auto_fill
```

Figure 9. Fortran subroutine `auto_fill`. This subroutine (1) calculates the α and β coefficients, (2) calculates the interface variables $c_{k,w}$, $c_{k,e}$, $\nabla c_{k,w}$, and $c\nabla_{k,e}$, (3) converts the change variables, c_k, and interface variables to dual numbers, (4) evaluates functions for the governing equations, `FLUX`, `RXN`, `ACCUM`, and boundary conditions, and (5) uses the results of the governing equations and boundary conditions to calculate the matrix coefficients $d_{i,k}$, $f_{i,k}$, $r_{i,k}$, and $g^{\circ}_{i,j}$.

```
subroutine ABDGXY(j)
    use user_input, only: N, NJ
    use GOV_EQNS
    use variables, only: A, B, D, G, X, Y
    implicit none
    integer :: j

    if (j == 1) then
      X = 0.0
      B = rj - (1.0 - alphaE)*fE + betaE*dE
      D(1:N,1:N) = -alphaE*fE - betaE*dE
      G = smG
      return
    end if

    if (j == NJ) then
      Y = 0.0
      A = (1.0 - alphaW)*fW - betaW*dW
      B = rj + betaW*dW + alphaW*fW
      G = smG
      return
    end if

    A = (1.0 - alphaW)*fW - betaW*dW
    B = rj + betaW*dW + alphaW*fW - (1.0 - alphaE)*fE + betaE*dE
    D(1:N,1:N) = -alphaE*fE - betaE*dE
    G = smG

    return
end subroutine ABDGXY
```

Figure 10. Subroutine `ABDGXY` uses the values of α, β, $d_{i,k}$, $f_{i,k}$, $r_{i,k}$, and $g^{\circ}_{i,j}$ calculated in `auto_fill` to evaluate the coefficients for the block tridiagonal matrices in `BAND` ($A^{\circ}_{i,k}$, $B^{\circ}_{i,k}$, $D^{\circ}_{i,k}$, $G^{\circ}_{i,j}$) as expressed by Eqs. 26–35.

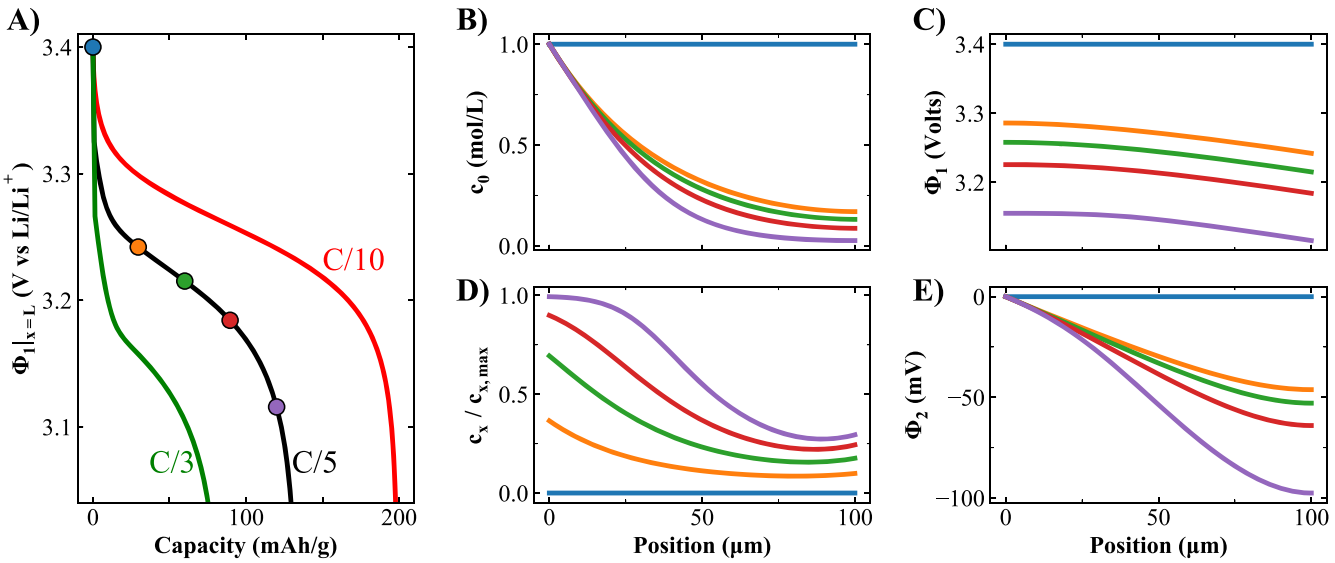

Figure 11. Simulated results for lithium insertion into a porous electrode. The equations used to simulate this phenomena are given in Table I. (A) The solid-state potential at the current collect ($\Phi_1|_{x=L}$) vs capacity for varying current rates: C/10, C/5, and C/3. (B)–(E) Time-varying profiles of the dependent variables for the C/5 data are displayed, where the line colors correspond to the marker color on the plot of $\Phi_1|_{x=L}$ vs capacity.

describe these complex phenomena. For the porous electrode problem discussed in this paper, the electrochemical equations that describe the open-circuit potential, U, the exchange current density, i_0, and the Butler-Volmer rate expression, i_n, were contained in a separate module, `echem_mod` (see Fig. 8). The code

The output from `auto_fill` (Fig. 9) is used by the subroutine `ABDGXY` (Fig. 10) to compute the matrix coefficients used by `BAND` (see Eqs. 26–35). Similar code to Fig. 10 can be found in Appendix 2 of J. Deliang Yang's thesis;[9] because this work is not available online, it has been reproduced here with some modifications to leverage modern Fortran's ability to perform array operations.

The simulation results of the model outlined in Tables I and II and linearized using dual number automatic differentiation are displayed in Fig. 11. The potential at the current collector, $\Phi_1|_{x=L}$ is displayed for three current rates C/10, C/5, C/3 (0.3, 0.6, 1.0 mA cm^{-2}); for C/5, profiles of the dependent variables, c_0, c_x, Φ_1, and Φ_2, are plotted at capacities of 0, 30, 60, 90, and 120 mAh g^{-1}. As discussed previously, dual number linearization is accurate to machine precision; therefore, the numerical results plotted in Fig. 11

Table III. Average and standard deviations of the computational run-times (in units of seconds) for the simulations displayed in Fig. 11. Code was compiled using `gfortran -fdefault-real-8 -O3` (compiler: GFortran, 8-byte real numbers, and all possible optimizations for speed), and run 30 times at each C-rate.

C-Rate	Conventional `BAND`	Using Dual Numbers
C/10	45.638 ± 0.262	48.348 ± 0.400
C/5	15.785 ± 0.422	16.800 ± 0.514
C/3	6.093 ± 0.076	6.810 ± 0.603

are identical to the results achieved using the more conventional manual linearization process as outlined by Newman[8]. In terms of computational efficiency, the results in Table III illustrate that the dual number implementation takes about 5%–10% longer than the conventional implementation for this particular problem.

Conclusions

BAND is a useful and flexible numerical tool. Coupling BAND with dual number automatic differentiation does not inhibit its utility, but allows for numerical results to be achieved with significantly less user programming, without compromising numerical precision, and only a small loss in computational efficiency. The procedure using BAND, FVM, and dnadmod, written as the subroutine auto_fill, is quite useful and intuitive and has been used to simulate a physical scenario relevant to battery applications.

Appendix

There are two points to discuss with respect to Fig. A·1. The first, as discussed throughout the paper, the length of the *dx* array is 2*N*; this length is set programmatically as ndv = N*2. The second important point is that, while dnadmod is extensive, it is not exhaustive. The user may find that an intrinsic Fortran function is not included in dnadmod; fortunately extending the module to include additional functions is relatively straightforward. The user needs to define the dual elemental function, i.e. tanh_d. The input (u) and output (res) to this function will both be type(dual). The dual elemental function is composed of two parts: the real part (res%x), which is equal to the intrinsic function (res%x = tanh(u%x)), and a differential part, which parallels the form:

$$\partial[f(x)] = f'(x)\cdot\partial x$$

For tanh(x)

$$\partial[\tanh(x)] = \frac{1}{\cosh^2(x)}\cdot\partial x$$

which is written programmatically as

```
res%dx = u%dx*1.0/cosh(u%x)**2
```

To overload the intrinsic function, one creates an interface within dnadmod with the same name as the intrinsic function (interface tanh) and within that interface the module procedure for a dual number is used (module procedure tanh_d). The statement public tanh ensures that the overloaded function is accessible to all modules that use dnadmod. The interface should appear above the contains statement in dnadmod and the elemental function should appear below the contains statement.

It may be desirable for user-defined functions to produce dual outputs in some instances and real outputs in other cases. For example, when writing data it is often simpler to use a function with real inputs and outputs. In these cases, it may be useful to overload the user-defined functions. Overloading a user-defined function is also relatively simple through the use of an interface block. In Fig. A·2, an overloaded function exchange_current is created from a real function and a type(dual) function. The output of exchange_current will be real, when the inputs are real, and dual when the inputs are dual.

```
module dnadmod

    use user_input, only: N
    implicit none

    integer, PARAMETER :: ndv = N*2   ! cprev_vars and dcdx_vars

    ...

    !-------------------------------------------------
    ! Interfaces for intrinsic functions overloading
    !-------------------------------------------------

    ...

    public tanh
    interface tanh
        module procedure tanh_d ! obtain atanh of a dual number, elemental
    end interface

  contains

    ...

    !----------------------------------------
    ! TANH of dual numbers
    ! <res, dres> = tanh(<u, du>) = <tanh(u), 1.0/cosh(u)**2 * du>
    !----------------------------------------
    elemental function tanh_d(u) result(res)
        type(dual), intent(in) :: u
        type(dual) :: res

        res%x = tanh(u%x)
        res%dx = u%dx * 1.0/cosh(u%x)**2

    end function tanh_d

end module dnadmod
```

Figure A·1. Fortran code fragment illustrating how to overload an intrinsic function such as atanh through the module dnadmod. The ellipses indicate lines of code that were omitted.

```
module echem_mod
  use user_input, only: Rigc, Fconst, Temp, max_Li_conc, cbulk
  use dnadmod

  implicit none

  ! public open_circuit_potential
  interface exchange_current
      module procedure exchange_current_density_real ! obtain i_0 using real numbers
      module procedure exchange_current_density_dual ! obtain i_0 using dual numbers
  end interface

contains

  function exchange_current_density_real(c0, ci)                    Result(i_0)
    real                          :: i_0
    real, intent(in)              :: c0                 ! c0 - electrolyte conc
    real, intent(in)              :: ci                 ! ci - solid-state conc

    i_0 = Fconst*k_rxn * c0**alpha_a * ci**alpha_c * (max_Li_conc - ci)**alpha_a
  end function exchange_current_density_real


  function exchange_current_density_dual(c0, ci)                    Result(i_0)
    type(dual)                    :: i_0
    type(dual), intent(in)        :: c0                 ! c0 - electrolyte conc
    type(dual), intent(in)        :: ci                 ! ci - solid-state conc

    i_0 = Fconst*k_rxn * c0**alpha_a * ci**alpha_c * (max_Li_conc - ci)**alpha_a
  end function exchange_current_density_dual

end module echem_mod
```

Figure A·2. The figure contains code for a user-defined function, `exchange_current_real`, with `real` inputs and output and a user-defined function, `exchange_current_dual`, with `type(dual)` inputs and output. In addition the `interface` block creates an overloaded user-defined function `exchange_current` from the `real` and `type(dual)` functions.

One may contend that overloading user-defined functions is a disadvantage of this automatic differentiation process. However, upon inspection of the functions in A·2, it is apparent that the real and dual functions are identical expect for the declarations of the input and output variables; if desired, one could even write a script to systematically overload user-defined functions.

ORCID

Nicholas W. Brady https://orcid.org/0000-0001-7877-6704
Maarten Mees https://orcid.org/0000-0001-7217-5510
Philippe M. Vereecken https://orcid.org/0000-0003-4115-0075
Mohammadhosein Safari https://orcid.org/0000-0003-0633-731X

References

1. H. Koh and C. L. Magee, "A functional approach for studying technological progress: Application to information technology." *Technological Forecasting and Social Change*, **73**, 1061 (2006).
2. W. D. Nordhaus, "Two centuries of productivity growth in computing." *The Journal of Economic History*, **67**, 128 (2007).
3. N. W. Brady, C. A. Gould, and A. C. West, "Quantitative Parameter Estimation, Model Selection, and Variable Selection in Battery Science." *J. Electrochem. Soc.*, **167**, 013501 (2020).
4. W. Yu and M. Blair, "DNAD, a Simple Tool for Automatic Differentiation of Fortran Codes Using Dual Numbers." *Comput. Phys. Commun.*, **184**, 1446 (2013).
5. R. E. Spall and W. Yu, "Imbedded dual-number automatic differentiation for Computational Fluid Dynamics sensitivity analysis." *Journal of fluids engineering*, **135**, 014501 (2013).
6. S. Stamatiadis and S. Farantos, "AUTO_DERIV: Tool for automatic differentiation of a Fortran code." *Comput. Phys. Commun.*, **181**, 1818 (2010).
7. A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation* (SIAM, Philadelphia, PA) 2nd ed. (2008).
8. J. Newman and K. E. Thomas-Alyea, *Electrochemical Systems* (Wiley, Hoboken, New Jersey) 3rd ed. (2004).
9. J. Deliang Yang, "Numerical Computation for Electrochemical Systems Using Finite Volume Method." *Experimental and Numerical Investigation of Mass Transfer in Electrochemical Systems*, Columbia University (1997), Columbia University.
10. J. Newman, "Numerical solution of coupled, ordinary differential equations." *Industrial & Engineering Chemistry Fundamentals*, **7**, 514 (1968).
11. J. Newman, *Numerical Solution Of Coupled, Ordinary Differential Equations UCRL-17739*, University of California Lawrence Radiation Laboratory (1967).
12. R. E. White, "On Newman's numerical technique for solving boundary value problems." *Industrial & Engineering Chemistry Fundamentals*, **17**, 367 (1978).
13. S. V. Patankar, *Numerical Heat Transfer and Fluid Flow* (CRC Press, Boca Raton, FL) 1st ed. (1980).
14. C. Hirsch, *Numerical Computation of Internal and External Flows* (Elsevier, Amsterdam) Fundamentals of Numerical Discretization, 2nd ed., 1 (2007).
15. A. West and T. Fuller, "Influence of rib spacing in proton-exchange membrane electrode assemblies." *Journal of applied electrochemistry*, **26**, 557 (1996).
16. R. H. Pletcher, J. C. Tannehill, and D. Anderson, *Computational Fluid Mechanics and Heat Transfer* (CRC Press, Boca Raton, FL) 3rd ed. (2012).
17. J. H. Ferziger, M. Perić, and R. L. Street, *Computational Methods for Fluid Dynamics* (Springer, Berlin) 3rd ed. (2002).
18. T. Sauer, "Numerical Differentiation and Integration." *Numerical Analysis* (Pearson, Boston, MA) 2nd ed., 5, 243 (2012).
19. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, " Evaluation of Functions." *Numerical Recipes: The Art of Scientific Computing* (Cambridge University Press, Cambridge, MA) 3rd ed., 5, 229 (2007).
20. J. N. Lyness and C. B. Moler, "Numerical Differentiation of Analytic Functions." *SIAM Journal on Numerical Analysis*, **4**, 202 (1967).
21. W. Squire and G. Trapp, "Using Complex Variables to Estimate Derivatives of Real Functions." *SIAM Review*, **40**, 110 (1998).
22. Andreas Griewank and Andrea Walther, "Introduction." *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation* (SIAM, Berlin) 2nd ed., 1 (2008).
23. G. F. Corliss, "Applications of Differentiation Arithmetic." *Reliability in Computing, The Role of Interval Methods in Scientific Computing* (Academic Press, Inc., Boston, MA) p.127-148 (1988).
24. A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic Differentiation in Machine Learning: a Survey." *Journal of Machine Learning Research*, **18**, 1 (2018).
25. Louis B. Rall, "Perspectives on Automatic Differentiation: Past, Present, and Future?" *Automatic Differentiation: Applications, Theory, and Implementations* (Springer, Berlin) (2006).
26. J. Revels, M. Lubin, and T. Papamarkou, *Forward-Mode Automatic Differentiation in Julia* (2016), arXiv:1607.07892[cs.MS].
27. S. F. Walter and L. Lehmann, "Algorithmic differentiation in Python with AlgoPy." *Journal of Computational Science*, **4**, 334 (2013).
28. M. A. Patterson, M. Weinstein, and A. V. Rao, "An Efficient Overloaded Method for Computing Derivatives of Mathematical Functions in MATLAB." *ACM Trans. Math. Softw.*, **39**, 17 (2013).
29. James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake Vander{P} las, Skye Wanderman-{M}ilne, and Qiao Zhang, (2018), JAX: Composable Transformations of Python + NumPy programs, http://github.com/google/jax.
30. W. K. Clifford, "Preliminary Sketch of Biquaternions." *Proceedings of the London Mathematical Society*, **4**, 381 (1873).
31. A. T. Yang and F. Freudenstein, "Application of Dual-Number Quaternion Algebra to the Analysis of Spatial Mechanisms." *Journal of Applied Mechanics*, **31**, 300 (1964).
32. J. Angeles, "The Application of Dual Algebra to Kinematic Analysis." *Computational Methods in Mechanical Systems: Mechanism Analysis, Synthesis, and Optimization* (Springer, Berlin) 3 (1998).
33. John Newman, (1998), FORTRAN Programs for the Simulation of Electrochemical Systems, http://www.cchem.berkeley.edu/jsngrp/fortran.html.
34. Wenbin Yu and Kshitiz Swaroop, (2014), Dual Number Automatic Differentiation, https://cdmhub.org/resources/374.