Journal of Computational Physics 228 (2009) 5526–5538

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

A direct matrix method for computing analytical Jacobians of discretized nonlinear integro-differential equations

Kevin T. Chu *

Vitamin D, Inc., Menlo Park, CA 94025, United States
*Institute of High Performance Computing, A*STAR, Singapore, Singapore*

ARTICLE INFO

Article history:
Received 12 December 2008
Received in revised form 20 April 2009
Accepted 22 April 2009
Available online 3 May 2009

Keywords:
Analytical Jacobian
Numerical methods
Matrix calculus
Newton's method
Integro-differential equations

ABSTRACT

In this article, we present a simple direct matrix method for analytically computing the Jacobian of nonlinear algebraic equations that arise from the discretization of nonlinear integro-differential equations. The method is based on a formulation of the discretized equations in vector form using only matrix-vector products and component-wise operations. By applying simple matrix-based differentiation rules, the matrix form of the analytical Jacobian can be calculated with little more difficulty than required to compute derivatives in single-variable calculus. After describing the direct matrix method, we present numerical experiments demonstrating the computational performance of the method, discuss its connection to the Newton–Kantorovich method and apply it to illustrative 1D and 2D example problems from electrochemical transport.

1. Introduction

When numerically studying nonlinear integro-differential equations, it is often valuable to have knowledge of the Jacobian for the discretized equations. For example, the Jacobian (either exact or approximate) is required during each iteration of the classical formulation of Newton's method. Unfortunately, calculation of the Jacobian can be a time-consuming and error-prone procedure for both the computer *and* the scientific programmer.

In the past, limited processing power and memory inspired the development and use of many methods designed to avoid computation of the full Jacobian. While Jacobian-free methods, such as the Newton–Krylov method [32] and Arnoldi iteration [42], are beautiful and continue to be critical for very large computational science problems, solving scientifically significant problems that are relatively large is now possible on modern workstations and laptops even if full Jacobians are used (especially if high-order numerical schemes are employed). As a result, researchers whose primary focus is science (as opposed to numerics) may find it more time effective to use simpler, classical numerical schemes in order to avoid having to learn the nuances of Jacobian-free methods. In this context, it is beneficial to have an easy way to compute the Jacobian of the discretized equations.

In this article, we present a simple *direct matrix method* for calculating analytical Jacobians of discretized, nonlinear integro-differential equations. The direct matrix method produces the Jacobian for the discretized equations directly in matrix form *without* requiring calculation of individual matrix elements. The essential idea is to first write the discretized, integro-differential equation explicitly in terms of discrete operators (e.g. differentiation and quadrature matrices [7,21,41]) and then use simple *matrix*-based differentiation rules to calculate the Jacobian directly [9,11,12,14]. The key observation

* Address: Vitamin D, Inc., Menlo Park, CA 94025, United States.
E-mail addresses: ktchu@serendipityresearch.org, kevin.t.chu@gmail.com

0021-9991/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2009.04.031

underlying this approach is that there is a tremendous amount of structure in the nonlinear algebraic equations that arise from the discretization of nonlinear integro-differential equations. By taking advantage of this structure, the calculation of analytical Jacobians is reduced to nearly the level of complexity required to compute derivatives of scalar, single-variable functions.

Operator-based approaches for expressing and analyzing numerical methods for time-independent nonlinear differential equations have been implicitly used by the scientific computing community for decades, especially in the context of the Newton–Kantorovich and related methods [5,6,29]. However, a direct matrix approach for the discretized equations seems to have been first formally described in one place by Chen who presented a collection of rules formulated in terms of specially defined matrix products [9,11,12]. Chen used his formulation of the method both to solve nonlinear partial differential equations [11,13] and to develop several interesting theoretical results (e.g. stability analysis of numerical methods for nonlinear time-dependent problems). He also observed that when a nonlinear differential equation only has polynomial nonlinearities, there is a very close relationship between the discretized nonlinear differential equation and its Jacobian [9,10].

Mathematically, the present formulation of the direct matrix method is equivalent to Chen's approach. However, rather than introducing special matrix products, we rely solely on standard linear algebra operations augmented by componentwise operations (e.g. the Hadamard or Schur product [24]). In addition, we have chosen to use MATLAB notation in our formulation because of its prevalence in modern scientific computing. Working in MATLAB notation has the added benefit of making it almost trivial to translate the analytical calculations into working numerical code.[1]

Another feature of our formulation, which is also present to some extent in [9], is the emphasis on the analogy between calculation of Jacobians for discretized, nonlinear integro-differential equations and calculation of derivatives for scalar functions of a single-variable. To help strengthen the analogy with single-variable calculus, we organize the operations required to compute a Jacobian as a short list of simple differentiation rules.

This article is organized as follows. In the remainder of this section, we compare the direct matrix method with several common methods for computing Jacobians. In Section 2, we present the direct matrix method, including a discussion of its computational performance and its relation to the Newton–Kantorovich method. Finally, in Section 3, we demonstrate the application of the direct matrix method to two examples (one 1D and one 2D) from the field of electrochemical transport. Throughout our discussion, we will focus solely on collocation methods where the continuous and discrete forms of the integro-differential equation have essentially the same structure. However, it is important to recognize that the direct matrix method can also be used for Galerkin methods by applying it directly to the weak-form of the problem.

1.1. Comparison with common methods for computing Jacobians

One common approach for obtaining the Jacobian of a discretized, nonlinear integro-differential equation is to compute it numerically using finite differences of the grid function or expansion coefficient values [27,28]. Unfortunately, numerical computation of the Jacobian can be time consuming for some problems. Depending on the numerical method, it might be possible to reduce the computational cost of a numerical Jacobian by taking advantage of the sparsity pattern in the Jacobian [18,19,28], but determining the sparsity pattern can be complicated for nontrivial problems.

As an example, consider the Poisson–Nernst–Planck equations for electrochemical transport [2,16,37]:

$$\frac{\partial c_+}{\partial t} = \nabla \cdot (\nabla c_+ + c_+ \nabla \phi) \tag{1}$$

$$\frac{\partial c_-}{\partial t} = \nabla \cdot (\nabla c_- - c_- \nabla \phi) \tag{2}$$

$$\epsilon \nabla^2 \phi = -(c_+ - c_-), \tag{3}$$

where $c_\pm$ and ϕ are the unknowns and ϵ is a physical parameter. The first two of these are the Nernst–Planck equations for ion transport and are simply conservation laws for cations and anions [34,37]. The last equation is the Poisson equation [26], which provides closure for the Nernst–Planck equations (1)–(3) form a nonlinear parabolic system of partial differential equations, which suggests that an implicit time-stepping scheme may be appropriate to numerically solve the equations. Unfortunately, an implicit scheme for this set of equations requires that we solve a nonlinear system of equations for $c_\pm^{(n+1)}$ which depends on an auxiliary variable ϕ which is in turn related to $c_\pm^{(n+1)}$ through the Poisson equation. As a result, to numerically compute the Jacobian for the resulting nonlinear system of equations for $c_\pm^{(n+1)}$, we must solve the Poisson equation for each perturbation to the current iterate of $c_\pm^{(n+1)}$. Therefore, for a pseudospectral discretization of (1)–(3) using N grid points, numerically computing the Jacobian requires $O(N^4)$ operations, which is much higher than the $O(N^2)$ elements in the Jacobian.[2]

Using an analytical Jacobian is one way to avoid the computational cost associated with numerical Jacobians. In principle, it is straightforward to derive the analytical Jacobian for the system of algebraic equations that arises when a nonlinear integro-differential equation is discretized. Index notation (also known as tensor notation) is perhaps the most common tech-

[1] With today's powerful desktop and laptop computers, MATLAB is quite capable of handling production-level research problems.

[2] While low-order discretization of the equations do not show this same disparity in the computation time and the number of elements (requiring $O(N^2)$ operations for $O(N^2)$ elements), they typically require many more grid points to produce an accurate solution.

nique used to calculate analytical Jacobians. The basic idea behind the index notation method is to write the discretized form of the differential equation using index notation and then use tensor calculus to compute individual matrix elements in the Jacobian. For example, for a finite-difference or pseudospectral discretization, the discretized equations can be written in the form:

$$F_i(u_1, u_2, \ldots, u_N) = 0 \tag{4}$$

for $i = 1, 2, \ldots, N$ where u_i and F_i are the value of the solution and the discretized differential equation at the ith grid point (or more generally, the ith collocation point). Boundary conditions are included in this formulation by using the discretized boundary conditions (rather than the integro-differential equation) at grid points on the boundary (or immediately adjacent to if no grid points reside on the boundary).[3] The ijth element of the Jacobian, $\mathbf{J}$, for (4) is simply the partial derivative of F_i with respect to $u_j : \mathbf{J}_{ij} = \partial F_i / \partial u_j$. While simple and straightforward, index notation suffers from the disadvantage of being somewhat tedious and error-prone. The main challenge in using index notation is keeping track of all of the indices when writing out and computing partial derivatives of the discrete equations.

Automatic differentiation [23,25] offers an important alternative when exact Jacobians for the discretized equations are desired. Because it generates code for computing the Jacobian directly from the code used to evaluate the residual, automatic differentiation completely eliminates the possibility of human error when deriving the exact Jacobian and implementing it in code. Recent developments have made automatic differentiation available in several common programming languages (including MATLAB [38]). While useful, automatic differentiation still takes some effort to use and may not always generate the most compact, efficient code. Active development in this area will certainly continue to improve the usability of automatic differentiation software and the performance of generated code.

The direct matrix method has several advantages over the methods discussed in this section. First, the direct matrix method yields a more accurate Jacobian than finite differences and generally in less time (see Section 2.4). Second, because the method is based on simple differentiation rules, the calculation is straightforward and less prone to error than the index notation approach. The differentiation rules also make it easier to calculate the Jacobian for differential equations which depend on auxiliary variables, such as (1)–(3). From a programming perspective, calculation of the Jacobian directly in matrix form facilitates implementation of numerical methods for nonlinear problems in languages that have built-in support for matrix and vector operations (e.g. MATLAB and Fortran 95). Finally, having the Jacobian available in matrix form can be useful for analyzing properties of numerical methods [9]. While it may be possible to convert the element-wise representation of the Jacobian derived using index notation for simple problems, this step can be challenging for more complex problems.[4]

2. The direct matrix method

There are two basic ideas underlying the direct matrix method for calculating analytical Jacobians of discretized, integro-differential equations. First, rather than writing the discretized, integro-differential equations at each of the collocation points in terms of individual elements of the solution vector, we write the entire system of equations as a single vector equation expressed explicitly in terms of matrix-vector products and component-wise multiplication (e.g. Hadamard products). Second, the analytical Jacobian for the discretized system of equations is computed directly in matrix form by using simple differentiation rules that are reminiscent of those used to compute derivatives in single-variable calculus. In this section, we present both of these ideas in detail. Towards the end of the section, we comment on the computational performance of the direct matrix method and its relationship to the Newton–Kantorovich method [7].

As we shall see, the direct matrix method is simply a formulation of well-known mathematical results into a convenient notation targeted towards numerical methods for integro-differential equations. However, it is important to recognize that the power of the method is precisely the simplicity of the notation, not its theoretical foundations.

2.1. Matrix-vector representation of discretized equations

Writing the discretized, nonlinear integro-differential equation explicitly in terms of basic linear algebra and component-wise algebraic operations is the initial step of the direct matrix method. Because of the similarities in the structure between the discrete and continuous forms of the equations, the procedure is very straightforward. First, convert all differentiation and integration operators into their discrete analogues. Since both of these operations are linear, they become multiplication of vectors representing grid functions by differentiation and quadrature matrices, respectively:

$$\frac{du}{dx} \to D * \hat{u} \tag{5}$$

$$\int u dx \to Q * \hat{u}, \tag{6}$$

where the hat accent indicates a discretized field variable and D and Q are the differentiation and quadrature matrices associated with the choice of computational grid.

[3] Care must be exercised when imposing boundary conditions, especially when using pseudospectral methods [22].

[4] Interestingly, the conversion from element-wise to matrix representation of the Jacobian often reveals the close relationship between the Jacobian for the discrete equations and the underlying structure of the original integro-differential equation.

Next, convert all point-wise algebraic operations and function evaluations in the continuous equations to component-wise algebraic operations and function evaluations in the discrete equations. Some examples of the conversion process include:

$$u\frac{dv}{dx} \rightarrow \hat{u} \cdot *(D * \hat{v}) \tag{7}$$

$$\sin(u) \rightarrow \sin(\hat{u}) \tag{8}$$

$$u^2 \rightarrow \hat{u} \cdot \wedge 2 \tag{9}$$

$$\exp(u) \rightarrow \exp(\hat{u}). \tag{10}$$

In these examples, we have adopted the MATLAB convention of using *.op* to represent component-wise application of the *op* operation. Also, note that we have abused notation for component-wise function evaluations – $f(\hat{u})$ represents the vector

$$(f(\hat{u}_1), f(\hat{u}_2), \ldots, f(\hat{u}_N)) \tag{11}$$

not an arbitrary vector function of the entire solution vector $\hat{u}$. Throughout our discussion, we will indicate component-wise and general functions of $\hat{u}$ by using lowercase and uppercase variables, respectively.

It is important to emphasize that the matrix-vector representation is not restricted to scalar field equations or functions of a single-variable. Handling vector equations is simple; vector equations may be treated as systems of equations and vector operations may be expressed in component-wise form. Equations in multiple space dimensions may be handled by using choosing an ordering for the grid points and defining differential/integral operators that are consistent with that ordering. For example, computations on structured grids typically use a lexicographic ordering for the grid points and define differential/integral operators using Kronecker products [41].

2.1.1. Boundary conditions in multiple space dimensions

Imposing complicated boundary conditions for problems in multiple space dimensions can be challenging. To simplify this process, it is often convenient to include discretized boundary conditions in the matrix-vector representation by first breaking the full differentiation and integration matrices into multiple components. Each component is defined by the grid points that it contributes to and the grid points it receives contributions from. For instance, it may be convenient to decompose a differentiation matrix into four mutually exclusive components that: (1) use interior points and contribute to interior points, (2) use interior points and contribute to boundary points, (3) use boundary points and contribute to interior points and (4) use boundary points and contribute to boundary points. Splitting the differentiation and integration matrices into separate components can be helpful when computing residuals and Jacobians for the different types of grid points in the computational domain.

Deriving these components is straightforward using zeroth-order restriction and prolongation matrices [8]. A zeroth-order restriction matrix is a matrix of zeros and ones which extracts a desired subset of elements from a vector. A zeroth-order prolongation matrix is also a matrix of zeros and ones, but it injects the elements of a restricted vector into a desired subset of the elements of a full-length vector. If $\{i_1, i_2, \ldots, i_m\}$ are the (flattened) indices of a subset of points from N grid points, then the associated restriction matrix, R, would be an $m \times N$ matrix, R, with ones at the positions $(1, i_1), (2, i_2), \ldots, (m, i_m)$. The associated prolongation matrix, P, that injects a vector of length m into the positions $\{i_1, i_2, \ldots, i_m\}$ of a vector of length N is simply the transpose of R: $P = R^T$.

To derive the differentiation matrix that uses values from grid points A and contributes to grid points B, we pre-multiply by the restriction matrix for the set B and post-multiply by the prolongation matrix for set A:

$$D_{A\rightarrow B} = R_B * D * P_A = R_B * D * R_A^T. \tag{12}$$

Restricted integration matrices are derived in exactly the same manner.

2.2. Simple differentiation rules for computing exact Jacobians

Once the continuous equations have been put in a discretized form that is expressed explicitly in terms of matrix-vector products and component-wise operations, the analytical Jacobian for the discretized equations can be calculated by applying a few simple *matrix*-based differentiation rules. Because the differentiation rules are expressed completely in matrix form without any reference to individual elements in the Jacobian matrix, they allow us to compute the Jacobian directly in matrix form. Furthermore, the direct matrix method emphasizes the close connection between the continuous and discretized problem, which makes it easier for application-oriented researchers to derive Jacobians for numerical or analytical purposes.

In this section, we list these differentiation rules, which are essentially results from multivariate and matrix calculus applied specifically to the structure of discretized integro-differential equations. More precisely, they are the Fréchet derivatives [7,33,40] of common expressions that appear in discretized nonlinear integro-differential equations. It is worth reiterating that the value of these rules is their simplicity and systematic formulation.

2.2.1. Matrix-vector product rule

The Jacobian of a matrix-vector product (which corresponds to a linear operator acting on a function in the continuous equations) is just the matrix itself:

$$\frac{\partial}{\partial \hat{u}}(A * \hat{u}) = A. \tag{13}$$

For example, the Jacobian of the discretized derivative of u, $D * \hat{u}$, is just D.

2.2.2. Diagonal rule

The Jacobian of a component-wise function f of a grid function $\hat{u}$ is a diagonal matrix with diagonal entries given by $f'(\hat{u})$:

$$\frac{\partial f(\hat{u})}{\partial \hat{u}} = \mathrm{diag}(f'(\hat{u})). \tag{14}$$

In essence, the diagonal rule is a way to use matrix notation to represent the fact that the differential in the ith component of $f(\hat{u})$ only depends on the change in the ith component of $\hat{u}$ and is given by $\delta f(\hat{u}_i) = f'(\hat{u}_i)\delta\hat{u}_i$. As an example, the Jacobian of $\sin(\hat{u})$ is $\mathrm{diag}(\cos(\hat{u}))$.

2.2.3. Chain rules

The Jacobian of a matrix A times an arbitrary function, F, of *all* of the components of $\hat{u}$ is A times the Jacobian of F:

$$\frac{\partial}{\partial \hat{u}}(A * F(\hat{u})) = A * \frac{\partial F}{\partial \hat{u}} \tag{15}$$

Similarly, the Jacobian of a function, $F(\hat{u})$, when its argument is a matrix A times the grid function $\hat{u}$ is the Jacobian of F evaluated at $A * \hat{u}$ times A:

$$\frac{\partial}{\partial \hat{u}} F(A * \hat{u}) = \left[\frac{\partial F}{\partial \hat{u}}(A * \hat{u})\right] * A \tag{16}$$

These rules are simply the chain rules for vector fields from multivariate calculus [1].

For the special but common case when F is a component-wise function $F(\hat{u}) = f(\hat{u})$, (15) reduces to A times the diagonal matrix with $f'(\hat{u})$ on the diagonal:

$$\frac{\partial}{\partial \hat{u}}(A * f(\hat{u})) = A * \mathrm{diag}(f'(\hat{u})) \tag{17}$$

and (16) reduces to the diagonal matrix with $f'(A * \hat{u})$ on the diagonal times the matrix A:

$$\frac{\partial}{\partial \hat{u}} f(A * \hat{u}) = \mathrm{diag}(f'(A * \hat{u})) * A \tag{18}$$

2.2.4. Product rule

To compute the Jacobian of a component-wise product of general functions F and G of a grid function $\hat{u}$, we use the product rule:

$$\frac{\partial}{\partial \hat{u}}(F(\hat{u}) \cdot * G(\hat{u})) = \mathrm{diag}(G(\hat{u})) * \frac{\partial F}{\partial \hat{u}} + \mathrm{diag}(F(\hat{u})) * \frac{\partial G}{\partial \hat{u}}. \tag{19}$$

The derivation of the product rule follows from the expression for the variation of the ith component of $F(\hat{u})G(\hat{u})$

$$\delta(F_i(\hat{u})G_i(\hat{u})) = G_i(\hat{u})\frac{\partial F_i}{\partial \hat{u}}\delta\hat{u} + F_i(\hat{u})\frac{\partial G_i}{\partial \hat{u}}\delta\hat{u} = \left(G_i(\hat{u})\frac{\partial F_i}{\partial \hat{u}} + F_i(\hat{u})\frac{\partial G_i}{\partial \hat{u}}\right)\delta\hat{u}, \tag{20}$$

which yields the Jacobian

$$\begin{bmatrix} G_1(\hat{u})(\partial F_1/\partial \hat{u}) \\ G_2(\hat{u})(\partial F_2/\partial \dot{u}) \\ \vdots \\ G_N(\hat{u})(\partial F_N/\partial \hat{u}) \end{bmatrix} + \begin{bmatrix} F_1(\hat{u})(\partial G_1/\partial \hat{u}) \\ F_2(\hat{u})(\partial G_2/\partial \dot{u}) \\ \vdots \\ F_N(\hat{u})(\partial G_N/\partial \hat{u}) \end{bmatrix} = \mathrm{diag}(G(\hat{u})) * \frac{\partial F}{\partial \hat{u}} + \mathrm{diag}(F(\hat{u})) * \frac{\partial G}{\partial \hat{u}} \tag{21}$$

2.3. Example Jacobian calculations

To demonstrate the simplicity of the direct matrix method, let us calculate the Jacobian for the discretized form of the nonlinear function

$$f(u) = e^{2u}\frac{du}{dx}. \tag{22}$$

Converting this function to discrete form, we obtain

$$f(\hat{u}) = (e \cdot \wedge(2\hat{u})) \cdot * (D * \hat{u}). \tag{23}$$

Using the product rule (19), we find that the Jacobian is given by

$$J = \mathrm{diag}(D * \hat{u}) * \frac{\partial}{\partial \hat{u}}(e \cdot \wedge(2\hat{u})) + \mathrm{diag}(e \cdot \wedge(2\hat{u})) * \frac{\partial (D * \hat{u})}{\partial \hat{u}}. \tag{24}$$

Then applying the diagonal rule (14) and the matrix-vector product rule (13), we find that

$$J = 2\mathrm{diag}(D * \hat{u}) * \mathrm{diag}(e \cdot \wedge(2\hat{u})) + \mathrm{diag}(e \cdot \wedge(2\hat{u})) * D, \tag{25}$$

which can be simplified to

$$J = 2\mathrm{diag}((D * \hat{u}) \cdot * (e \cdot \wedge(2\hat{u}))) + \mathrm{diag}(e \cdot \wedge(2\hat{u})) * D \tag{26}$$

by observing that $\mathrm{diag}(\hat{u} \cdot * \hat{v}) = \mathrm{diag}(\hat{u}) * \mathrm{diag}(\hat{v})$.

The direct matrix method also makes it easy to calculate the Jacobian when auxiliary variables are present. As an example, we compute the Jacobian for the nonlinear algebraic equations that arise when solving the one-dimensional version of (1)–(3) using a simple backwards Euler discretization in time. Using the direct matrix approach for the spatial discretization, the nonlinear algebraic equations for $\hat{c}_+^{(n+1)}$ and $\hat{c}_-^{(n+1)}$ that need to be solved at each time step are:

$$\hat{c}_+^{(n+1)} - \Delta t(L * \hat{c}_+^{(n+1)} + D * (\hat{c}_+^{(n+1)} \cdot * (D * \hat{\phi}))) - \hat{c}_+^{(n)} = 0 \tag{27}$$

$$\hat{c}_-^{(n+1)} - \Delta t(L * \hat{c}_-^{(n+1)} - D * (\hat{c}_-^{(n+1)} \cdot * (D * \hat{\phi}))) - \hat{c}_-^{(n)} = 0 \tag{28}$$

$$\epsilon L\hat{\phi} + (\hat{c}_+^{(n+1)} - \hat{c}_-^{(n+1)}) = 0 \tag{29}$$

where $\hat{c}_\pm^{(n)}$ are the concentrations at the current time step, L is the discretized Laplacian operator and Δt is the time step size. It is important to mention that several of the rows in (29) will typically be replaced to impose the discretized form of the boundary conditions for ϕ. For illustrative purposes, let us suppose that we have simple Dirichlet boundary conditions for ϕ. In this situation, (29) is only imposed at interior grid points [41].

Using the simple differentiation rules from the previous section, the Jacobian of (27) with respect to $\hat{c}_+^{(n+1)}$ is

$$I - \Delta t\left(L + D * \mathrm{diag}(D * \hat{\phi}) + D * \mathrm{diag}(\hat{c}_+^{(n+1)}) * D * \frac{\partial \hat{\phi}}{\partial \hat{c}_+^{(n+1)}}\right), \tag{30}$$

where I is the identity matrix and $(\partial\hat{\phi}/\partial\hat{c}_+^{(n+1)})$ is the Jacobian of $\hat{\phi}$ with respect to $\hat{c}_+^{(n+1)}$. To eliminate $(\partial\hat{\phi}/\partial\hat{c}_+^{(n+1)})$ from this expression, we simply apply the differentiation rules to (29) with two rows eliminated for the boundary conditions and solve for the interior portion of $(\partial\hat{\phi}/\partial\hat{c}_+^{(n+1)})$:

$$\left(\frac{\partial \hat{\phi}}{\partial \hat{c}_+^{(n+1)}}\right)_{int} = -\frac{1}{\epsilon}(L)_{int}^{-1}, \tag{31}$$

where $(L)_{int}$ is the submatrix of L that remains when all of the columns and rows corresponding to boundary grid points have been removed. Since the boundary values of $\hat{\phi}$ are fixed and the values of $\hat{c}_+^{(n+1)}$ at the boundaries do not affect the potential in the interior, the full Jacobian $(\partial\hat{\phi}/\partial\hat{c}_+^{(n+1)})$ is given by

$$\frac{\partial \hat{\phi}}{\partial \hat{c}_+^{(n+1)}} = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & -\frac{1}{\epsilon}(L)_{int}^{-1} & \vdots \\ 0 & \cdots & 0 \end{bmatrix}, \tag{32}$$

where we have assumed that the first and last grid points correspond to boundary points. It is important to recognize that the form for $(\partial\hat{\phi}/\partial\hat{c}_+^{(n+1)})$ in (32) is specific to problems with Dirichlet boundary conditions for ϕ. For other boundary conditions, the inversion of the equation for $(\partial\hat{\phi}/\partial\hat{c}_+^{(n+1)})$ generally leads to different forms for the Jacobian.

The Jacobian for (27) can now be explicitly computed by substituting (32) into (30). The similar expression for the Jacobian of (28) is obtained using an analogous procedure. Using the direct matrix approach, we have reduced the calculation of the Jacobian to $O(N^3)$ (cost of matrix-inversion and matrix–matrix multiplies) compared to the $O(N^4)$ cost for computing a numerical Jacobian for high-order spatial discretizations. It is worth pointing out that in this example, the Jacobian for the concentrations does *not* depend explicitly on ϕ because the Poisson equation is linear. Thus, in contrast to a numerical Jacobian calculation, there is no need to solve for ϕ in order to compute the analytical Jacobians for (27) and (28). For general problems, the Jacobian may depend on the auxiliary variable, so it might be necessary to solve the constraint equation. However, because only one solve for the auxiliary variables is required with the direct matrix method, the cost of computing the Jacobian is still dramatically reduced compared to using finite differences.

2.4. Computational performance

In general, using the direct matrix method to compute a Jacobian is faster than calculating a numerical Jacobian. As mentioned in the previous section, the performance difference is expected to be large when auxiliary variables are involved in the expression of the residual. However, the direct matrix method yields higher performance even for problems where the residual is relatively simple.

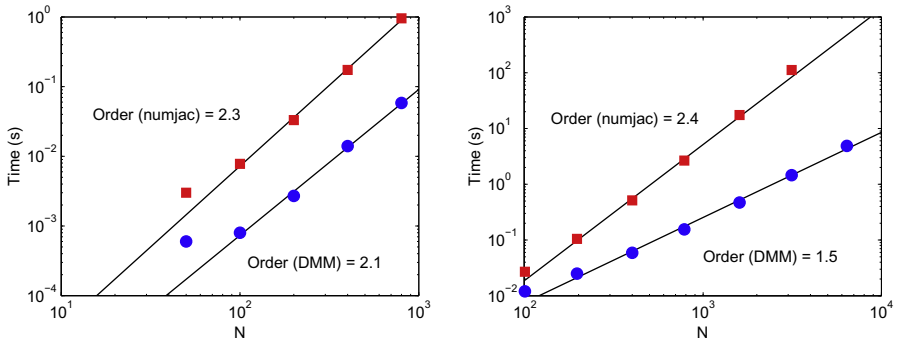

Fig. 1. Comparison of the computational performance of direct matrix method (circles) and numerical Jacobian computed using MATLAB `numjac()` function (squares) for the 1D electrochemical thin-film problem (Section 3.1) and the 2D metal colloid sphere problem (Section 3.2). In these figures, N is the total number of grid points in the computational mesh. The data for these graphs were generated on a 2.4 GHz MacBook Pro with 4 GB of RAM. Note that for the 2D problem, no `numjac()` data is available for the largest grid size because the system ran out of memory.

Fig. 1 compares the performance of the direct matrix method against the MATLAB `numjac()` function for the two example problems discussed in Section 3. As we can see, the direct matrix method is at least an order of magnitude faster for both the 1D and 2D problems. For the 2D problem, the direct matrix method also shows superior scaling with the grid size. To ensure a fair comparison, we vectorized the residual calculation to minimize the number of function calls required by `numjac()` and avoided the use of unnecessary matrix multiplications,[5] which benefited both methods. Matrix–matrix multiplications are especially detrimental for the direct matrix method because they can worsen the scaling of the Jacobian construction time with grid size to the point where the numerical Jacobian is faster to compute. For instance, in the left graph in Fig. 1, a Jacobian computed using the direct matrix method with explicit matrix–matrix multiplications take $O(N^3)$ time, which negates the performance benefits of the method compared with a `numjac()` implementation before N even reaches 1000.

In addition to avoiding matrix–matrix multiplication, it is important to use sparse matrices when possible. For problems in more than one space dimension that are discretized on structured grids, sparse matrices are produced when Kronecker products with identity matrices are used to construct differentiation matrices even if the 1D differentiation matrices are dense. The benefit of using sparse matrices when computing the Jacobian using the direct matrix method is demonstrated in the right graph in Fig. 1. For that 2D problem, there are $O(N^{1/2})$ grid points in each coordinate direction. Because a pseudo-spectral discretization is used, the partial differential operators all have $O((N^{1/2})^2 N^{1/2}) = O(N^{3/2})$ elements, which explains the scaling observed in the figure.

2.5. Relationship to the Newton–Kantorovich method

The direct matrix method for computing the Jacobian of discretized integro-differential equations is closely related to the calculation of the Fréchet derivative[6] used in the Newton–Kantorovich method [7,29] (also known as quasilinearization [20]). The basic idea behind solving nonlinear integro-differential equations using the Newton–Kantorovich method is to carry out Newton's method in *function space*. For each Newton iteration, we compute the Fréchet derivative of the integro-differential equation in function space and numerically solve the resulting *linear* integro-differential equation for the correction to the current iterate of the solution. Essentially, the Newton–Kantorovich method reverses the order of (1) discretization of the continuous problem and (2) Newton iteration. Because the equations to be solved during each Newton iteration is linear, there is no need to compute a Jacobian of the discretized equations.

An important feature of the Newton–Kantorovich method is that the numerical discretization used to solve the linearized equation during each Newton iteration can, in principle, be completely independent of the discretization used to compute the residual of the nonlinear integro-differential equation. This freedom can affect the convergence rate of the method depending on the degree to which the discretized form of the linearized problem approximates the Jacobian of the discretized residual equation.

Because the direct matrix method begins with a discrete equation possessing the same mathematical structure as the continuous residual equation, it produces a Jacobian that is a discrete analogue of the Fréchet derivative for the continuous integro-differential equation. Unlike the Newton–Kantorovich method, however, the direct matrix method produces the unique Jacobian associated with the particular choice of discretization for the residual of the nonlinear integro-differential equation. The freedom to independently choose the numerical discretizations for the residual equation and the Fréchet derivative is not present in the direct matrix method. As a result, given a numerical discretization for the residual equation,

[5] For example, we express matrix-vector products of the form $\mathrm{diag}(\hat{u}) * \hat{v}$ as component-wise multiplication of two grid functions $\hat{u} .* \hat{v}$.

[6] Recall that the Fréchet derivative for nonlinear functionals is the generalization of the Jacobian for nonlinear functions over finite-dimensional spaces [7,33,40]. For intuition, Ortega and Rheinboldt provide a nice discussion of Fréchet derivatives in the context of finite-dimensional spaces [36].

the direct matrix method can be viewed as a way to generate the optimal discretization for the linearized equation that arises during each Newton iteration of the Newton–Kantorovich method.

3. Applications

In this section, we apply the direct matrix method to two real-world problems from the field of electrochemical transport (which has recently seen a resurgence of interest [2–4,15,30,31,35,39]). As we shall see, these equations, while complicated, are straightforward to solve using Newton's method and the direct matrix method.

3.1. Electrochemical thin-films

Analysis of 1D electrochemical systems leads to an example of a nonlinear integro-differential equation. For steady-state electrochemical thin-films made up of a dilute solution of symmetric binary electrolyte with faradaic reactions at the surfaces of the thin-film [2,15], the electric field, E, satisfies the second-order differential equation.

$$\epsilon^2\left(\frac{d^2E}{dx^2}-\frac{1}{2}E^3\right)-\frac{1}{4}(c_0+j(x+1))E-\frac{j}{4}=0 \tag{33}$$

on the domain $(-1,1)$ subject to boundary conditions that represent the kinetics of electrode reactions

$$-k_c(c(1)+\rho(1))+j_r-j=0 \tag{34}$$

$$k_c(c(-1)+\rho(-1))-j_r-j=0, \tag{35}$$

where c is the average ion concentration, ρ is the charge density, j is the current density flowing through the thin-film and ϵ, k_c and j_r are physical constants. The average ion concentration, $c(x)$ and charge density, $\rho(x)$, may be written in terms of the electric field as

$$c(x)=c_0+j(x+1)+2\epsilon^2E^2,\quad \rho(x)=4\epsilon^2\frac{dE}{dx}. \tag{36}$$

Finally, c_0 arises from the imposition of an integral constraint on the ion concentration and is given by

$$c_0=(1-j)+\epsilon^2\left[2E(1)-2E(-1)-\int_{-1}^{1}E^2dx\right]. \tag{37}$$

We can solve this set of equations via Newton's method using a systematic application of the direct matrix method. To discretize the equations, we use a pseudospectral method based on the Chebyshev grid on the interval $[-1,1]$. The differentiation matrix, D, for this computational grid is just the standard differentiation matrix for the Chebyshev grid [7,21,41]. For numerical integration, we use the Clenshaw–Curtis quadrature weights [41], which we denote by the row vector w. The quadrature weights are used to construct a quadrature matrix, Q, which is the analogue of the differentiation matrix: $Q=[w^T,w^T,\ldots,w^T]^T$. When a grid function f is multiplied by Q, the result is a vector where all entries are equal to the numerical approximation of the integral of f.

With these discrete operators, we can put (33) into matrix-vector form:

$$\epsilon^2\left(D^2*\hat{E}-\frac{1}{2}\hat{E}\cdot\wedge 3\right)-\frac{1}{4}(\hat{C}_0+j(x+1))\cdot*\hat{E}-\frac{j}{4}=0 \tag{38}$$

with

$$\hat{C}_0=(1-j)+\epsilon^2(2\hat{E}_1-2\hat{E}_N-Q*(\hat{E}\cdot\wedge 2)), \tag{39}$$

where we have chosen to order the indices so that $x_1=1$ and $x_N=-1$. The boundary conditions are imposed by replacing the discrete equations corresponding to x_1 and x_N with

$$-k_c(\hat{c}_0+2j+\epsilon^2(2\hat{E}_1^2+4D_1*\hat{E}))+j_r-j=0 \tag{40}$$

$$k_c(\hat{c}_0+\epsilon^2(2\hat{E}_N^2+4D_N*\hat{E}))-j_r-j=0, \tag{41}$$

where D_1 and D_N are the rows of the differentiation matrix corresponding to x_1 and x_N, respectively and $\hat{c}_0$ is a single component of $\hat{C}_0$.

The Jacobians for the left-hand side of these discrete equations are now easily computed. Applying the differentiation rules from Section 2, the Jacobian for the interior grid points is

$$J_{int}=\epsilon^2\left(D^2-\frac{3}{2}\mathrm{diag}(\hat{E}\cdot\wedge 2)\right)-\frac{1}{4}\mathrm{diag}(\hat{C}_0+j(x+1))-\frac{1}{4}\mathrm{diag}(\hat{E})*\frac{\partial\hat{C}_0}{\partial\hat{E}} \tag{42}$$

with

$$\frac{\partial \hat{C}_0}{\partial \hat{E}} = \epsilon^2 \left(\begin{bmatrix} 2 & 0 & \cdots & 0 & -2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 2 & 0 & \cdots & 0 & -2 \end{bmatrix} - 2Q * \mathrm{diag}(\hat{E}) \right) \tag{43}$$

The Jacobian for the discretized boundary conditions are similarly calculated:

$$J_1 = -k_c \left(\frac{\partial \hat{c}_0}{\partial \hat{E}} + 4\epsilon^2 [E_1 \ 0 \ \ldots \ 0] + 4\epsilon^2 D_1 \right) \tag{44}$$

$$J_N = k_c \left(\frac{\partial \hat{c}_0}{\partial \hat{E}} + 4\epsilon^2 [0 \ \ldots \ 0 \ E_N] + 4\epsilon^2 D_N \right), \tag{45}$$

where

$$\frac{\partial \hat{c}_0}{\partial \hat{E}} = \epsilon^2 ([2 \ 0 \ \ldots \ 0 \ -2] - 2w * \mathrm{diag}(\hat{E})). \tag{46}$$

Now that we have explicitly computed all of the components for Newton's method, it is straightforward to write a program to solve the nonlinear integro-differential equations for electrochemical thin-films. The MATLAB code (see supplementary online material) for solving is relatively short and runs quickly. One special issue that arises for this problem is that continuation methods [7] are required to obtain good initial iterates for the Newton iteration at high current densities. Fig. 2 shows the numerical solution of (33)–(35) computed using 100 grid points for various values of j. As expected, we observe geometric convergence with respect to the number of grid points (see Fig. 3). Notice that at higher current densities, we see slower convergence rates due to the presence of greater structure in the solution.

From the perspective of computational performance, the above formulation of the Jacobian is suboptimal because it includes a matrix–matrix multiply in (42) that can be avoided. To reduce the time required to compute the Jacobian, the key observation is that each row of $\frac{\partial \hat{C}_0}{\partial \hat{E}}$ is equal $\frac{\partial \hat{c}_0}{\partial \hat{E}}$. Therefore, $\mathrm{diag}(\hat{E}) * \frac{\partial \hat{C}_0}{\partial \hat{E}}$ is more efficiently computed as the Kronecker product of $\hat{E}$ and $\frac{\partial \hat{c}_0}{\partial \hat{E}}$. The evaluation of the residual can also be improved by recognizing that all of the elements of $\hat{C}_0$ are equal to $\hat{c}_0$, but this optimization has a far smaller impact than the reformulation of the Jacobian.

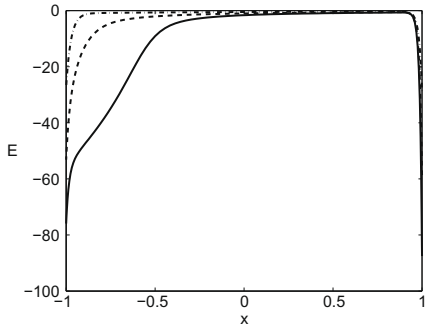

Fig. 2. Solution of electrochemical thin-film Eqs. (33)–(35) computed using 100 grid points with $\epsilon = 0.01, k_c = 10$ and $j_r = 10$ for $j = 1.5$ (solid), $j = 1.0$ (dash) and $j = 0.5$ (dot-dash).

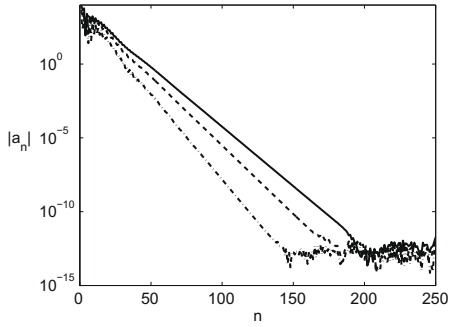

Fig. 3. Plots of the absolute value of the spectral coefficients, a_n, for the numerical solution of the electrochemical thin-film equations with $j = 1.5$ (solid), $j = 1.0$ (dash) and $j = 0.5$ (dot-dash) as a function of the basis function degree. Note that the convergence rate as a function of number of grid points is geometric. The spectral coefficients were calculated from a numerical solution generated using 250 grid points to make the roundoff plateau more apparent.

3.2. Double layer charging of metal colloid sphere at high applied electric fields

Analysis of double layer charging for colloid systems subject to applied electric fields gives rise to nonlinear differential equations in multiple space dimensions with complicated boundary conditions. We will use this problem to demonstrate the ease with which the direct matrix method handles complicated boundary conditions in more than one space dimension. In particular, it illustrates how decomposition of differential operators into interior and boundary pieces can simplify computation of the Jacobian for the discretized equations.

In the electroneutral limit [2,34,37], the steady-state governing equations for systems composed of symmetric binary electrolyte are [16]

$$\nabla^2 c = 0 \tag{47}$$

$$\nabla \cdot (c\nabla\phi) = 0, \tag{48}$$

where c is the average ion concentration and ϕ is the electric potential. For metal colloid surfaces, the appropriate boundary conditions are [16,17]

$$0 = \epsilon\nabla_s \cdot (q\nabla_s \ln c + w\nabla_s\phi) - c\frac{\partial\phi}{\partial n} \tag{49}$$

$$0 = \epsilon\nabla_s \cdot (w\nabla_s \ln c + q\nabla_s\phi) - \frac{\partial c}{\partial n} \tag{50}$$

$$q = -2\sqrt{c}\sinh(\zeta/2) \tag{51}$$

$$w = 4\sqrt{c}\sinh^2(\zeta/4) \tag{52}$$

$$v - \phi = \zeta + 2\delta\sqrt{c}\sinh(\zeta/2) \tag{53}$$

where q, w and ζ are surface field variables related to the boundary layer structure and v and δ are physical parameters of the system. As a model problem, we solve these equations for a metal colloid sphere subjected to a uniform applied electric field of strength E in the z-direction. To avoid infinite values of the electric potential, the numerical model is formulated in terms of $\psi \equiv \phi + Ez$, the deviation of the electric potential from that of the uniform applied field.

While this problem may seem daunting, it is straightforward to obtain a solution numerically by using Newton's method with an analytical Jacobian computed using the direct matrix method. Taking advantage of azimuthal symmetry, we discretize the equations in spherical coordinates on a 2D pseudospectral grid that is the tensor product of grids in the radial and polar angle directions. We use a shifted semi-infinite rational Chebyshev grid [7] in the radial direction and a uniformly spaced grid for the polar angle direction. The required differentiation matrices are constructed using Kronecker products and the boundary conditions are handled using restriction and prolongation matrices as discussed in Section 2.1.1.

To facilitate the formulation of the matrix-vector representation of the equations, let us fix our notation. Let D_r and D_θ be the radial and angular contributions to the discrete divergence operator, G_r and G_θ be the radial and angular components of the discrete gradient operator and L be the discrete Laplacian operator. Also, let n and s subscripts denote normal and tangential derivative operators at the surface of the sphere.

For the purpose of discussion (and implementation), it is convenient to decompose the discrete differential operators into pieces that correspond to contributions from finite and infinite grid points. For example, L can be decomposed into L^f and L^∞ which respectively account for the contributions to the Laplacian operator from finite and infinite grid points; that is, $L * \hat{c} = L^f * \hat{c}_f + L^\infty * \hat{c}_\infty$, where $\hat{c}_f$ and $\hat{c}_\infty$ are the concentration values at finite and infinite grid points respectively. Similarly, to impose the boundary conditions, we use derivative operators that act only on surface values. Surface operators and surface field values will be denoted with superscripts s and subscripts s, respectively. Finally, to refer to values at interior grid points (i.e., finite grid points that are *not* on the surface of the sphere), we use the subscript i.

In this notation, the discretized form of the bulk Eqs. (47) and (48) are given by

$$0 = F_1 \equiv L^f * \hat{c}_f + L^\infty * c_\infty \tag{54}$$

$$0 = F_2 \equiv D_r^f * [\hat{c}_f \cdot *(G_r^f * \hat{\psi}_f - E\cos\theta)] - D_r^\infty * (c_\infty \cdot * E\cos\theta) + D_\theta^f * [\hat{c}_f \cdot * (G_\theta^f * \hat{\psi}_f + E\sin\theta)] + D_\theta^\infty * (c_\infty \cdot * E\sin\theta). \tag{55}$$

In these equations, the unknowns are the values of the c and ψ at finite grid points; values at infinity are specified by the boundary conditions and so are known quantities (which is why c_∞ does not have a hat accent and $\psi_\infty = 0$ does not show up at all). In discretized form, the boundary conditions on the surface of the sphere are

$$0 = H_1 \equiv \epsilon D_s * [\hat{q} \cdot *(G^s * \ln \hat{c}_s) + \hat{w} \cdot * (G^s * \hat{\psi}_s - G^s * E\cos\theta)] - c_s \cdot * (G_n^f * \hat{\psi}_f + E\cos\theta) \tag{56}$$

$$0 = H_2 \equiv \epsilon D_s * [\hat{w} \cdot *(G^s * \ln \hat{c}_s) + \hat{q} \cdot * (G^s * \hat{\psi}_s - G^s * E\cos\theta)] - (G_n^f * \hat{c}_f + G_n^\infty * c_\infty). \tag{57}$$

Closure for these equations is given by using (51) and (52) to relate $\hat{q}$ and $\hat{w}$ to the zeta-potential and using (53) to compute the zeta-potential from ϕ and $\hat{c}_s$.

The direct matrix method makes it straightforward to derive the analytical Jacobian for the system of Eqs. (51)–(57). The derivatives of F_1 and F_2 with respect to the unknowns $\hat{c}_f$ and $\hat{\psi}_f$ are easily calculated:

$$\frac{\partial F_1}{\partial \hat{c}_f} = L^f \tag{58}$$

$$\frac{\partial F_1}{\partial \hat{\psi}_f} = 0 \tag{59}$$

$$\frac{\partial F_2}{\partial \hat{c}_f} = D_r^f * \mathrm{diag}(G_r^f * \hat{\psi}_f - E\cos\theta) + D_\theta^f * \mathrm{diag}(G_r^f * \hat{\psi}_f + E\sin\theta) \tag{60}$$

$$\frac{\partial F_2}{\partial \hat{\psi}_f} = D_r^f * \mathrm{diag}(\hat{c}_f) * G_r^f + D_\theta^f * \mathrm{diag}(\hat{c}_f) * G_\theta^f \tag{61}$$

The derivatives for the discretized boundary conditions are more complicated because $\hat{q}, \hat{w}$ and $\hat{c}_s$ implicitly depend on the unknown variables and because surface grid points must be treated differently than interior grid points. However, a systematic application of the differentiation rules in Section 2.2 yields the analytical Jacobian directly in matrix form:

$$\begin{aligned}\frac{\partial H_1}{\partial \hat{c}_s} = {} & \frac{\epsilon}{2} D_s * \mathrm{diag}(q \cdot / c_s \cdot * (G^s * \ln c_s)) - \epsilon D_s * \mathrm{diag}\left(\sqrt{c_s} \cdot * \cosh(\zeta/2) \cdot * \frac{\partial \zeta}{\partial c_s} \cdot * (G^s * \ln c_s)\right) \\ & + \epsilon D_s * \mathrm{diag}(q) * G^s * \mathrm{diag}(1 \cdot / c_s) + \frac{\epsilon}{2} D_s * \mathrm{diag}(w \cdot / c_s \cdot * (G^s * \psi_s - G^s * E\cos\theta)) \\ & + \epsilon D_s * \mathrm{diag}\left(\sqrt{c_s} \cdot * \sinh(\zeta/2) \cdot * \frac{\partial \zeta}{\partial c_s} \cdot * (G^s * \psi_s - G^s * E\cos\theta)\right) - \mathrm{diag}\left(G_n^f * \psi_f + E\cos\theta\right)\end{aligned} \tag{62}$$

$$\frac{\partial H_1}{\partial c_i} = 0 \tag{63}$$

$$\begin{aligned}\frac{\partial H_1}{\partial \psi_s} = {} & -\epsilon D_s * \mathrm{diag}\left(\sqrt{c_s} \cdot * \cosh(\zeta/2) \cdot * \frac{\partial \zeta}{\partial \psi_s} \cdot * (G^s * \ln c_s)\right) + \epsilon D_s * \mathrm{diag}(w) * G^s \\ & + \epsilon D_s * \mathrm{diag}\left(\sqrt{c_s} \cdot * \sinh(\zeta/2) \cdot * \frac{\partial \zeta}{\partial \psi_s} \cdot * (G^s * \psi_s - G^s * E\cos\theta)\right) - \mathrm{diag}(c_s) * G_n^s\end{aligned} \tag{64}$$

$$\frac{\partial H_1}{\partial \psi_i} = -\mathrm{diag}(c_s) G_n^i \tag{65}$$

$$\begin{aligned}\frac{\partial H_2}{\partial \hat{c}_s} = {} & \frac{\epsilon}{2} D_s * \mathrm{diag}(w \cdot / c_s \cdot * (G^s * \ln c_s)) + \epsilon D_s * \mathrm{diag}\left(\sqrt{c_s} \cdot * \sinh(\zeta/2) \cdot * \frac{\partial \zeta}{\partial c_s} \cdot * (G^s * \ln c_s)\right) \\ & + \epsilon D_s * \mathrm{diag}(w) * G^s * \mathrm{diag}(1 \cdot / c_s) + \frac{\epsilon}{2} D_s * \mathrm{diag}(q \cdot / c_s \cdot * (G^s * \psi_s - G^s * E\cos\theta)) \\ & - \epsilon D_s * \mathrm{diag}\left(\sqrt{c_s} \cdot * \cosh(\zeta/2) \cdot * \frac{\partial \zeta}{\partial c_s} \cdot * (G^s * \psi_s - G^s * E\cos\theta)\right) - G_n^s\end{aligned} \tag{66}$$

$$\frac{\partial H_2}{\partial c_i} = -G_n^i \tag{67}$$

$$\begin{aligned}\frac{\partial H_2}{\partial \psi_s} = {} & \epsilon D_s * \mathrm{diag}\left(\sqrt{c_s} \cdot * \sinh(\zeta/2) \cdot * \frac{\partial \zeta}{\partial \psi_s} \cdot * (G^s * \ln c_s)\right) + \epsilon D_s * \mathrm{diag}(q) * G^s \\ & - \epsilon D_s * \mathrm{diag}\left(\sqrt{c_s} \cdot * \cosh(\zeta/2) \cdot * \frac{\partial \zeta}{\partial \psi_s} \cdot * (G^s \psi_s - G^s * E\cos\theta)\right)\end{aligned} \tag{68}$$

$$\frac{\partial H_2}{\partial \psi_i} = 0 \tag{69}$$

where

$$\frac{\partial \zeta}{\partial \psi_s} = -\frac{1}{1 + \delta\sqrt{c_s}\cosh(\zeta/2)} \tag{70}$$

$$\frac{\partial \zeta}{\partial c_s} = -\frac{\delta \sinh(\zeta/2)}{\sqrt{c_s}[1 + \delta\sqrt{c_s}\cosh(\zeta/2)]}. \tag{71}$$

The Jacobian for the system of equations is obtained by assembling these pieces:

$$J = \begin{bmatrix} \frac{\partial F_1}{\partial \hat{c}} & \frac{\partial F_1}{\partial \hat{\psi}} \\ \frac{\partial F_2}{\partial \hat{c}} & \frac{\partial F_2}{\partial \hat{\psi}} \\ \frac{\partial H_1}{\partial \hat{c}} & \frac{\partial H_1}{\partial \hat{\psi}} \\ \frac{\partial H_2}{\partial \hat{c}} & \frac{\partial H_2}{\partial \hat{\psi}} \end{bmatrix}, \tag{72}$$

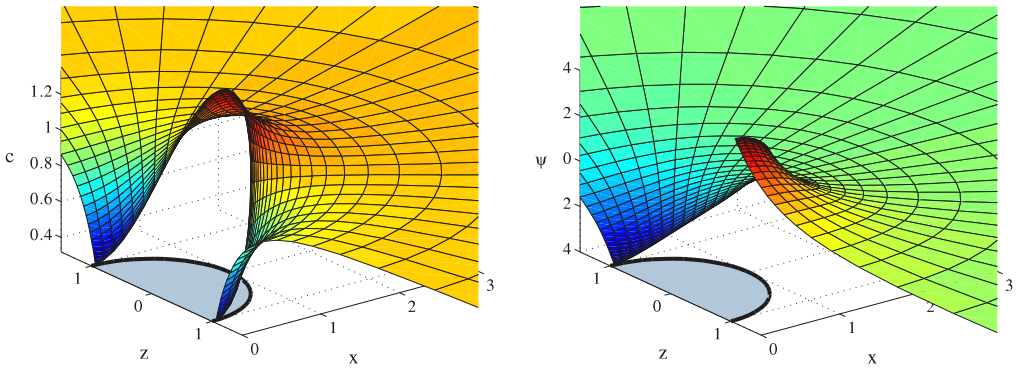

Fig. 4. Solution of Eqs. (47)–(53) computed using 30 grid points in both the radial and polar angle directions with $E = 10$, $v = 0, \epsilon = 0.01$ and $\delta = 1$. The scale parameter is set to 0.5 for the shifted rational Chebyshev grid.

where the Jacobians for H_1 and H_2 are constructed from (62)–(69) using restriction operators. For instance,

$$\frac{\partial H_1}{\partial \hat{c}} = \frac{\partial H_1}{\partial \hat{c}_s} * R_s + \frac{\partial H_1}{\partial \hat{c}_i} * R_i, \tag{73}$$

where R_s and R_i are restriction operators for surface and interior grid points, respectively. While the formulas may look complicated to program, the use of MATLAB notation makes it trivial to implement a solution in MATLAB (see supplementary online material).

Fig. 4 shows numerical solutions obtained using the above residual and Jacobian formulas. As for the electrochemical thin-film example, continuation is required to obtain good initial iterates for the Newton iteration at high values of the applied electric fields. Using pseudospectral grids and the analytical Jacobian, the solution is obtained very quickly, requiring only a few Newton iterations for each continuation stage (and less than a minute of computation time on a 2.4 GHz MacBook Pro).

4. Conclusions

In this article, we have presented a direct matrix method for calculating analytical Jacobians for discretized, nonlinear integro-differential equations. Because this method is based on simple matrix-based differentiation rules, it is less tedious and less error-prone than other approaches for computing analytical Jacobians. The power of the method originates precisely from the convenience of the notation and its formulation as a simple set of differentiation rules. These features of the method make it particularly useful for application-oriented computational scientists. Furthermore, because the direct matrix method yields the Jacobian in matrix form, it is straightforward to implement the analytical Jacobian in code using languages that support vectorized computation (e.g. MATLAB and Fortran 95).

One interesting possibility that the direct matrix method presents is development of high-level automatic differentiation tool for discretized nonlinear integro-differential equations. In contrast to traditional automatic differentiation methods [23,25] which operate at the level of individual scalar operations, automatic differentiation methods based on the direct matrix method would operate on the discrete differential operators associated with the continuous differential equation. Such an automatic differentiation tool could be useful for completely eliminating the need for a researcher to compute the Jacobian of discretized nonlinear integro-differential equations by hand.

Acknowledgments

The author gratefully acknowledges the support of the Department of Energy through the Computational Science Graduate Fellowship (CSGF) Program provided under Grant number DE-FG02-97ER25308, Vitamin D, Inc., and the Institute for High-Performance Computing (IHPC) in Singapore. The author thanks B. Kim, P. Fok and J.P. Boyd for many helpful discussions and suggestions.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.jcp.2009.04.031.

References

[1] T.M. Apostol, Calculus, vol. II, John Wiley and Sons, Inc., 1969.
[2] M.Z. Bazant, K.T. Chu, B.J. Bayly, Current–voltage relations for electrochemical thin films, SIAM J. Appl. Math. 65 (2005) 1463–1484.

[3] M.Z. Bazant, T.M. Squires, Induced-charge electrokinetic phenomena: theory and microfluidic applications, Phys. Rev. Lett. 92 (2004) 066101.
[4] P.M. Biesheuvel, A.A. Franco, M.Z. Bazant, Diffuse charge effects in fuel cell membranes, J. Electrochem. Soc. 156 (2009) B225–B233.
[5] J.P. Boyd, An analytical and numerical study of the two-dimensional Bratu equation, J. Sci. Comput. 1 (1986) 183–206.
[6] J.P. Boyd, Solitons from sine waves: analytical and numerical methods for non-integrable solitary and cnoidal waves, Physica 21D (1986) 227–246.
[7] J.P. Boyd, Chebyshev and Fourier Spectral Methods, third ed., Dover Publications, Inc., Mineola, NY, 2001.
[8] W.L. Briggs, V.E. Henson, S.F. McCormick, A Multigrid Tutorial, SIAM, 2000.
[9] W. Chen, Jacobian matrix: a bridge between linear and nonlinear polynomial-only problems, 1999, arXiv:cs/9904006v1.
[10] W. Chen, Relationship theorem between nonlinear polynomial equation and the corresponding Jacobian matrix, Int. J. Nonlinear Sci. Numer. Simul. 1 (2000) 5–14.
[11] W. Chen, C. Shu, W. He, T. Zhong, The application of special matrix product to differential quadrature solution of geometrically nonlinear bending of orthotropic rectangular plates, Comput. Struct. 74 (2000) 65–76.
[12] W. Chen, T. Zhong, The study on the nonlinear computations of the DQ and DC methods, Numer. Meth. Partial Diff. Eqn. 13 (1997) 57–75.
[13] W. Chen, T. Zhong, Y. Yu, Applying special matrix product to nonlinear numerical computations, J. Appl. Comp. Math. 12 (1998) 51–58.
[14] K.T. Chu, Asymptotic Analysis of Extreme Electrochemical Transport, Ph.D. Thesis, Massachusetts Institute of Technology, Department of Mathematics, 2005.
[15] K.T. Chu, M.Z. Bazant, Electrochemical thin films at and above the classical limiting current, SIAM J. Appl. Math. 65 (2005) 1485–1505.
[16] K.T. Chu, M.Z. Bazant, Nonlinear electrochemical relaxation around conductors, Phys. Rev. E 74 (2006) 011501.
[17] K.T. Chu, M.Z. Bazant, Surface conservation laws at microscopically diffuse interfaces, J. Colloid Interf. Sci. 315 (2007) 319–329.
[18] T.F. Coleman, J.J. Moré, Estimation of sparse Jacobian matrices and graph coloring problems, SIAM J. Numer. Anal. 20 (1983) 187–209.
[19] A.R. Curtis, M.J.D. Powell, J.K. Reid, On the estimations of sparse Jacobian matrices, J. Inst. Math. Appl. 13 (1974) 117–119.
[20] P. Deuflhard, Newton Methods for Nonlinear Problems, Springer, Berlin, Germany, 2004.
[21] B. Fornberg, A Practical Guide to Pseudospectral Methods, Cambridge University Press, New York, NY, 1998.
[22] B. Fornberg, A pseudospectral fictitious point method for high order initial-boundary value problems, SIAM J. Sci. Comput. 28 (2006) 1716–1729.
[23] A. Griewank, A. Walther, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, SIAM, Philadelphia, PA, 2008.
[24] R.A. Horn, C.R. Johnson, Topics in Matrix Analysis, Cambridge University Press, 1991.
[25] P. Hovland, B. Norris, Argonne National Laboratory Computational Differentiation Project, 2006, <http://www-fp.mcs.anl.gov/autodiff/>.
[26] J.D. Jackson, Classical Electrodynamics, John Wiley and Sons Inc., 1998.
[27] C.T. Kelley, Iterative Methods for Solving Linear and Nonlinear Equations, SIAM, Philadelphia, PA, 1995.
[28] C.T. Kelley, Solving Nonlinear Equations with Newton's Method, SIAM, Philadelphia, PA, 2003.
[29] C.T. Kelley, E.W. Sachs, A quasi-Newton method for elliptic boundary value problems, SIAM J. Numer. Anal. 24 (1987) 516–531.
[30] M.S. Kilic, M.Z. Bazant, A. Ajdari, Steric effects in the dynamics of electrolytes at large applied voltages: I. Double-layer charging, Phys. Rev. E 75 (2007) 021502.
[31] M.S. Kilic, M.Z. Bazant, A. Ajdari, Steric effects in the dynamics of electrolytes at large applied voltages: II. Modified Nernst–Planck equations, Phys. Rev. E 75 (2007) 021503.
[32] D.A. Knoll, D.E. Keyes, Jacobian-free Newton–Krylov methods: a survey of approaches and applications, J. Comput. Phys. 193 (2004) 357–397.
[33] A.N. Michel, C.J. Herget, Applied Algebra and Functional Analysis, Dover Publications, Inc., 1981.
[34] J. Newman, Electrochemical Systems, second ed., Prentice-Hall, Inc., Englewood Cliffs, NJ, 1991.
[35] L.H. Olesen, H. Bruus, A. Ajdari, AC electrokinetic micropumps: the effect of geometrical confinement Faradaic current injection and nonlinear surface capacitance, Phys. Rev. E 73 (2006) 056313.
[36] J.M. Ortega, W.C. Rheinboldt, Iterative Solution of Nonlinear Equations in Several Variables, Academic Press, New York, NY, 1970.
[37] I. Rubinstein, Electro-Diffusion of Ions, SIAM Studies in Applied Mathematics, SIAM, Philadelphia, PA, 1990.
[38] L.F. Shampine, R. Ketzscher, S.A. Forth, Using AD to solve BVPs in MATLAB, ACM T. Math. Software 31 (2005) 1–16.
[39] T.M. Squires, M.Z. Bazant, Induced-charge electro-osmosis, J. Fluid Mech. 509 (2004) 217–252.
[40] A.E. Taylor, The differential: nineteenth and twentieth century developments, Arch. Hist. Exact Sci. 12 (1974) 355–383.
[41] L.N. Trefethen, Spectral Methods in MATLAB, SIAM, Philadelphia, PA, 2000.
[42] L.N. Trefethen, D. Bau, Numerical Linear Algebra, SIAM, Philadelphia, PA, 1997.