Scalable Higher-order Nonlinear Solvers via Higher-order Automatic Differentiation

Songchen Tan[1] Keming Miao[1] Alan Edelman[1] Christopher Rackauckas[1]

[1]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA,
{songchen,xbtmkmmm,edelman,crackauc}@mit.edu

Abstract

This paper demonstrates new methods and implementations of nonlinear solvers with higher-order of convergence, which is achieved by efficiently computing higher-order derivatives. Instead of computing full derivatives, which could be expensive, we compute directional derivatives with Taylor-mode automatic differentiation. We first implement Householder's method with arbitrary order for one variable, and investigate the trade-off between computational cost and convergence order. We find that the second-order variant, i.e., Halley's method, to offer the best trade-off between computational cost and convergence rate, and further generalize Halley's method to systems of nonlinear equations and demonstrate that it can scale efficiently to large-scale problems. We further apply Halley's method on solving large-scale ill-conditioned nonlinear problems, as well as solving nonlinear equations inside stiff ODE solvers, and demonstrate that it could outperform Newton's method.

Keywords: nonlinear solvers, automatic differentiation, Householder's method, Halley's method

1 Introduction

Solving nonlinear equations accurately and efficiently is crucial in many disciplines of science, where nonlinear equations can emerge either directly from the models, or as an intermediate step of solving ordinary differential equations (ODEs)(Byrne and Hindmarsh 1987), partial differential equations (PDEs)(Ames 2014), differential-algebraic equations (DAEs)(Kunkel 2006), and integral equations(Atkinson 1992).

Various methods are available to solve nonlinear equations, and the mostly used ones are Newton's method and its variants(Kelley 2003), which rely on the first-order derivative, i.e. the Jacobian, or its approximations. Newton's method is proved to have quadratic convergence under certain assumptions of the nonlinear function and initial condition(Galántai 2000).

There has been extensive research on improving the convergence order of nonlinear solvers, which can be categorized into two different approaches: (1) multi-step methods that evaluate nonlinear functions and/or derivatives at multiple points in each iteration(Potra 1984; X. Xiao and Yin 2015; Madhu and Jayaraman 2017; X.-Y. Xiao 2022; Singh and Sharma 2023); (2) methods that utilize higher-order derivatives of nonlinear functions(Alefeld 1981; A. Cuyt and Cruyssen 1983; A. A. M. Cuyt and Rall 1985; A. A. Cuyt 1982; K. I. Noor, M. A. Noor, and Momani 2007; Ogbereyivwe, Izevbizua, and Umar 2023; Sariman and Hashim 2020; Germani et al. 2006). Despite their higher convergence order, in order to be more efficient than Newton's method, they must be computationally efficient for each iteration. For example, some multi-step methods(Singh and Sharma 2023) evaluate the function and Jacobian at multiple points, but only invert the Jacobian at one point; in practice, this would mean that for a nonlinear function with n inputs and n outputs, the $O(n^3)$ factorization of the Jacobian, which is the bottleneck of Newton's method, only needs to be done once for each iteration. This ensures that these multi-step methods are not much more expensive than Newton's method for each iteration.

The same considerations also apply to methods of the second category, where higher-order derivatives introduce additional computation burden. It is commonly believed that computing the higher-order full derivative of a function with large input and output dimension n would be much more expensive than the computation of the function itself(Margossian 2019). As a result, they are often approximated rather than exactly computed(Sariman and Hashim 2020; Suleiman 2009; Albeanu 2008; Steihaug 2006), and it is still an open question whether nonlinear solvers can utilize higher-order derivative information for large-scale nonlinear problems in an exact way.

This paper demonstrates new methods and implementations that, instead of computing the full derivative, efficiently and exactly compute the higher-order directional derivative, which is sufficient for nonlinear solvers. This paper is organized as follows. We will first introduce the technique of Taylor-mode automatic differentiation (AD) for efficiently computing higher-order directional derivatives, and its implementation in Julia, in section 2. Then we will demonstrate that Taylor-mode AD can be used to efficiently implement Householder's method with arbitrary convergence order, in section 3. Finally, we demonstrate that a special case of Householder's method, the cubic-convergence Halley's method can scale elegantly to large-scale problems, and could outperform first-order methods by a significant margin, in section 4.

2 Taylor-mode automatic differentiation for higher-order directional derivatives

We first introduce the notation used in this paper. Let U be an open subset of $\mathbb{R}^n$, $f : U \to \mathbb{R}^m$ be a function that is sufficiently smooth. The derivative of f at a point $x \in \mathbb{R}^n$ is a linear operator $Df(x) : \mathbb{R}^n \to \mathbb{R}^m$ such that it maps a vector $v \in \mathbb{R}^n$ to $Df(x)[v]$, which is the directional derivative of f at x in the direction of v, also known as the Jacobian-vector product.

Similarly, for $p \in \mathbb{Z}^+ > 1$, the p-th order derivative of f at a point x is a multilinear operator $D^p f(x) : (\mathbb{R}^n, \cdots, \mathbb{R}^n) \to \mathbb{R}^m$ such that it maps a tuple of vectors $(v_1, \ldots, v_p)$ to $D^p f(x)[v_1, \cdots, v_p]$, the directional derivative of f at x in the direction of $(v_1, \ldots, v_p)$.

First-order forward-mode AD can be viewed as an algorithm to propagate directional derivative information through compositions of functions(Revels, Lubin, and Papamarkou 2016). For example, let $f(x) = g(h(x))$ to be a composite function for $\mathbb{R}^n \to \mathbb{R}^m$, and $x \in \mathbb{R}^n$ to be a specific point in its domain. Suppose that we already have $h_0 = h(x)$ to represent the primal output of h, and $h_1 = Dh(x)[v]$ representing the perturbation of h along some direction $v \in \mathbb{R}^n$. Now for the composite function, we want to know what is the perturbation of f along direction v; in other words, we want to know $f_1 = Df(x)[v]$ in addition to $f_0 = f(x)$. The answer to that is simply the chain rule:

- $f_0 = g(h_0)$
- $f_1 = Dg(h_0)[h_1]$

Similarly, the Taylor-mode AD can be viewed as an algorithm to propagate higher-order directional derivative information through compositions of functions. Suppose that we already have a Taylor coefficients for h at x along v:

$$(h_0, h_1, \ldots, h_p) = (h(x), Dh(x)[v], \ldots, D^p h(x)[v, \ldots, v])$$

and we want to know

$$(f_0, f_1, \ldots, f_p) = (f(x), Df(x)[v], \ldots, D^p f(x)[v, \ldots, v])$$

the answer to that is, in turn, the Faà di Bruno's formula:

- $f_0 = g(h_0)$
- $f_1 = Dg(h_0)[h_1]$
- $f_2 = D^2 g(h_0)[h_1, h_1] + Dg(h_0)[h_2]$
- $\ldots$

In our practical implementation of Taylor-mode AD, a pushforward rule is defined for every "simple" function, so that derivatives of any complicated functions that are composed of these simple functions can be computed automatically, either via operator-overloading or source-code transformation techniques. Our previous work (Tan 2023a) has shown that for any functions composed of elementary functions and arbitrary control flow, the p-th order directional derivative is at most $O(p^2)$ times more expensive than computing the function itself, in contrast to naively nesting first-order forward-mode AD which could be $O(\exp(p))$ times expensive(Bettencourt, Johnson, and Duvenaud 2022). We also provided a reference implementation of this method in the Julia language(Bezanson, Edelman, et al. 2017) using its multiple-dispatch mechanism(Bezanson, Bolewski, and Chen 2018), TaylorDiff.jl(Tan 2023b), featuring an automatic generation(Gowda, Ma, Cheli, et al. 2021) of higher-order pushforward rules from first-order pushforward rules defined in ChainRules.jl(White et al. 2023).

Below, we will demonstrate how to use Taylor-mode AD to implement higher-order nonlinear solvers. For all numerical experiments in this paper, we use the Julia language v1.11.2 on a single core of Intel Xeon Platinum 8260 CPU provided by the MIT SuperCloud system.

3 Efficient implementation of Householder's method

We begin by implementing Householder's method(Householder 1970) for solving nonlinear equations with one variable, i.e. $x \in \mathbb{R}$ in the nonlinear function $f(x)$. This does not immediately generalize to multiple variables, but it is useful to compare the behavior of solvers with different convergence order. The method is defined as follows: given an initial guess x_0, in each iteration we compute,

$$x_{n+1} = x_n + p\frac{(1/f)^{(p-1)}(x_n)}{(1/f)^{(p)}(x_n)}.$$

which requires derivatives up to p-th order and has convergence order $p+1$. Note that when $p = 1$, this method is equivalent to Newton's method; and when $p = 2$, this method is equivalent to Halley's method.

Given f and x_n, we can compute $f(x_n)$ and its derivatives up to p-th order using Taylor mode AD, and then invert this Taylor polynomial to get the value and derivatives of $1/f$ up to p-th order. Our analysis leads to the algorithm 1:

3.1 Cost-effectiveness for Householder's method with different orders.

We implement algorithm 1 as a part of SimpleNonlinearSolve.jl(Pal et al. 2024), which is a package of non-allocating nonlinear solvers that have low-overhead and could solve small nonlinear problems very efficiently. We make several univariate nonlinear functions composed of various elementary functions, and use different orders of Householder's method to solve $f(x) = 0$ given appropriate initial guess. Results are shown in fig. 1.

DOI 10.3384/ecp218861

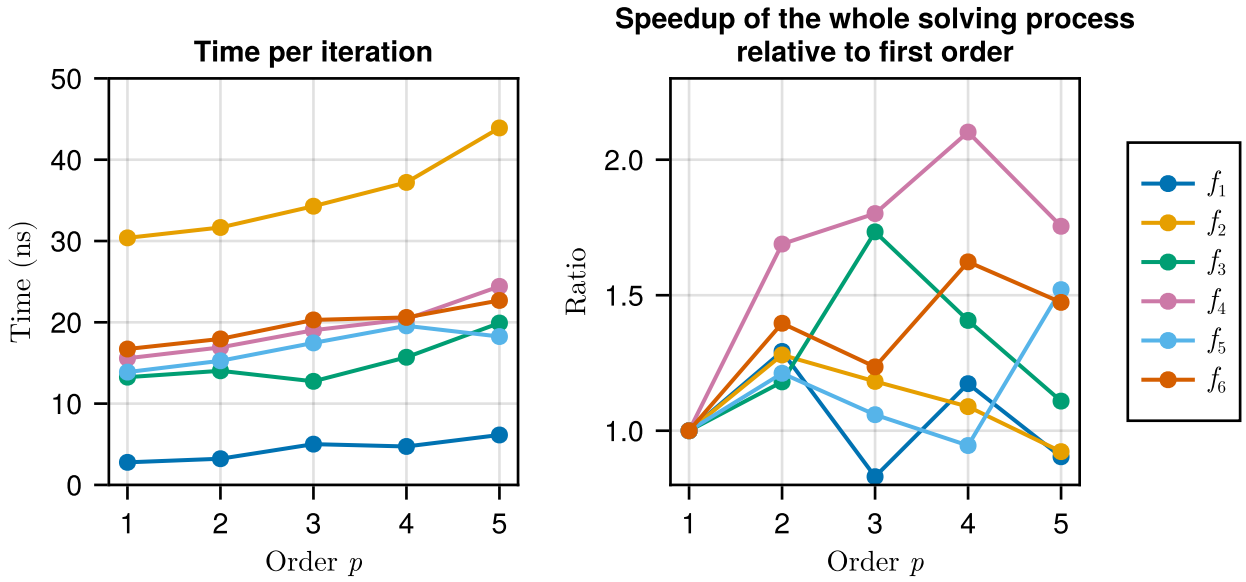

Figure 1. Cost-effectiveness for Householder's method with different orders. Left: the computation time per iteration for each nonlinear function and each Householder's method with different order. Right: the relative speedup for each function of the total computation time to solve the nonlinear equation, normalized by the $p = 1$ solver (Newton's method). The functions and initial conditions are: (1) $f_1(x) = x^2 - 2$, $x_0 = 1.0$; (2) $f_2(x) = \sqrt{x} - \pi$, $x_0 = 10.0$; (3) $f_3(x) = x - \exp(-x)$, $x_0 = 0.0$; (4) $f_4(x) = x^2 - 2^x$, $x_0 = 3.3$; (5) $f_5(x) = x + \sin(x) - 1$, $x_0 = 0.5$; (6) $f_6(x) = \log(x) + x$, $x_0 = 1.0$.

Algorithm 1 Householder's method implementation based on higher-order AD

Define f and initial value x; tolerance t; order p
while $|f(x)| > t$ **do**
  Initialize Taylor coefficients $(x, 1, 0, \cdots, 0)$
  Apply the pushforward rule of f to get $(f(x), f'(x), \cdots, f^{(p)}(x))$
  Apply the pushforward rule of $(\cdot)^{-1}$ to get $((1/f)(x), (1/f)'(x), \cdots, (1/f)^{(p)}(x))$
  Update $x := x + p\frac{(1/f)^{(p-1)}(x)}{(1/f)^{(p)}(x)}$.
end while
return x

For each of the nonlinear functions, the computation cost per iteration only slightly increases as the order p goes from 1 (Newton's method) to 5, but the number of iterations needed to converge could be cut down significantly by the higher-order methods. As the result, the total time needed to converge could be shorter for higher-order methods. We also notice that the most significant improvement is from the first order to the second order, and the improvement from the second order to an even higher order is less significant.

3.2 Generalization to multivariate functions

The theory of abstract rational approximation(A. Cuyt and Cruyssen 1983) gives a method to generalize Householder's method to multivariate functions. In the sections below, we will focus on implementing $p = 2$ case, i.e. Halley's method, with abstract rational approximation formulas, for multivariate functions. Based on the observations from simple univariate problems, we hypothesize that higher-order methods with $p \geq 3$ yield diminishing returns in practice.

4 Scalable and Efficient implementation of Halley's method

Halley's method ($p = 2$) for a multivariate real function $f(x) : \mathbb{R}^n \to \mathbb{R}^n$, derived from abstract rational approximation(A. Cuyt and Cruyssen 1983; A. A. M. Cuyt and Rall 1985), could be expressed as follows: given an initial guess x_0, in each iteration we compute

$$x_{n+1} = x_n + (a_n \odot a_n) \oslash (a_n + [Df(x_n)]^{-1} D^2 f(x_n)[a_n, a_n]/2) \quad (1)$$

where $\odot$ and $\oslash$ are element-wise multiplication and division, Df and $D^2 f$ are the Jacobian and Hessian of f, and a_n is the solution to $Df(x_n)a_n = -f(x_n)$. In the equation above, $D^2 f(x_n)a_n a_n$ can be computed easily using Taylor-mode AD, with just two times more expensive than the computation of $f(x_n)$. After the first linear solve for a_n, we need another linear solve to get $[Df(x_n)]^{-1} D^2 f(x_n)a_n a_n$. Among the two linear solves, the factorization of Jacobian could be reused, similar to the case of multi-step methods(Singh and Sharma 2023). Our analysis leads to algorithm 2:

Looking into the details of algorithm 2, we note that for

Algorithm 2 (Multivariate) Halley's method implementation based on higher-order AD

Define f and initial value x; tolerance t
while $\|f(x)\|_{\text{inf}} > t$ **do**
  Obtain via first-order AD the Jacobian $Df(x)$
  Solve a from $Df(x)[a] = -f(x)$
  Initialize Taylor coefficients $(x, a, 0)$
  Apply the pushforward rule of f to get $(f(x), Df(x)[a], D^2f(x)[a,a])$
  Solve b from $Df(x)[b] = D^2(x)[a,a]$
  Update $x := x + a \odot a \oslash (a + b/2)$
end while
return x

either dense or sparse Jacobian, the construction and factorization of $Df(x)$ (via LU or QR factorization) should be the bottleneck of solving, and these only need to be done once for the two linear solves. In other words, given the factorization, the additional linear solve for b can be computed in $O(n^2)$ time. Therefore, each step of Halley's method can be asymptotically as cheap as Newton's method, which makes it potentially faster than Newton's method.

4.1 Solving nonlinear problems with dense Jacobian

We first test algorithm 2 on nonlinear problems where the Jacobian of the nonlinear function $f(x) : \mathbb{R}^n \to \mathbb{R}^n$ is dense and unstructured. In this case, the factorization of the Jacobian is the bottleneck of the linear solve, and the cost of each factorization is $O(n^3)$. For defining such a problem, we consider the Chandrasekhar's H-function(Chandrasekhar 2013) which is the solution to an integral equation that arises in the study of radiative transfer in astrophysics. The equation is defined as follows:

$$H(\mu) = 1 + \mu H(\mu) \int_0^1 \frac{\Psi(\mu')H(\mu')}{\mu + \mu'} \mathrm{d}\mu' \tag{2}$$

where $H(\mu)$ is defined on $[0,1]$ and $\Psi(\mu)$ is an even polynomial. We consider the most simplified case, where Ψ is a constant. Upon discretization onto n grids for integration, the equation becomes a system of nonlinear equations of n variables, and since each equation depends on the value of H on all positions, the Jacobian of this nonlinear problem is inherently dense. The detailed form of the discretized nonlinear function is available as the 23rd problem in NonlinearProblemLibrary.jl(Pal et al. 2024) which the reader can refer to.

We solve this problem with different problem sizes $n = 4, 8, 16, 32, 64, 128$, and we compare the performance of (1) Newton's method, (2) Halley's method as implemented in algorithm 2, and (3) naive Halley's method implemented in a conventional way(A. A. M. Cuyt and Rall 1985) that computes the full Hessian and then contracts it with vectors. Each of them is solved to the default tolerance. The results are shown in fig. 2.

While the performances are similar for small problems, the naive Halley's method quickly becomes infeasible for large problems(Pal et al. 2024), while our implementation of Halley's method scales similarly as Newton's method. In addition, as the problem size gets larger, the advantage of Halley's method over Newton's method becomes more significant, as a result of fewer iterations needed to converge. For example, for $n = 128$, Halley's method is approximately 40% faster than Newton's method. Finally, more detailed work-precision diagrams comparing Halley's method and Newton's method, within a broader range of problem sizes ($n = 4, 16, 64, 256, 1024$), is shown in fig. 3.

4.2 Solving large-scale ill-conditioned nonlinear problems with sparse Jacobian

Nonlinear problems that appear in solving PDEs are often large, sparse and ill-conditioned(Ames 2014), imposing additional challenges for nonlinear solvers. Below, we will use a two-dimensional Brusselator reaction-diffusion PDE problem as an example to demonstrate the capability of Halley's method to handle these nonlinear problems in practical applications. The Brusselator is a model for auto-catalytic chemical systems that exhibit oscillations in time domain and pattern formation in space domain(Prigogine and Lefever 1968). Let functions $u(x,y,t)$ and $v(x,y,t)$ be concentrations of substances that are defined on $(x,y) \in [0,1] \times [0,1]$ and $t \in (0, +\infty)$, satisfying the following differential equations,

$$\begin{aligned} u_t &= B + u^2 v - (A+1)u + \alpha \Delta u + f(x,y,t) \\ v_t &= Au - u^2 v + \alpha \Delta v \end{aligned} \tag{3}$$

where the source f is defined as

$$f(x,y,t) = \begin{cases} 5 & \text{if } (x-0.3)^2 + (y-0.6)^2 \le 0.1^2 \text{and } t \ge 1.1 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

and initial conditions,

$$\begin{aligned} u(x,y,0) &= 22(y(1-y))^{3/2} \\ v(x,y,0) &= 27(x(1-x))^{3/2} \end{aligned} \tag{5}$$

and periodic boundary conditions.

$$\begin{aligned} u(0,y,t) &= u(1,y,t) \\ u(x,0,t) &= u(x,1,t) \\ v(0,y,t) &= v(1,y,t) \\ v(x,0,t) &= v(x,1,t) \end{aligned} \tag{6}$$

In the following numerical experiments, the parameters are set to $A = 3.4$, $B = 1$, $\alpha = 10$.

We first consider the time-independent PDE problem, where for $t \gg 1.1$ we solve the steady-state equations $u_t = v_t = 0$ to get the long-time behavior of the spatial distribution of concentrations u and v. In order to apply nonlinear solvers, the spatial discretization of u and v is

DOI 10.3384/ecp218861

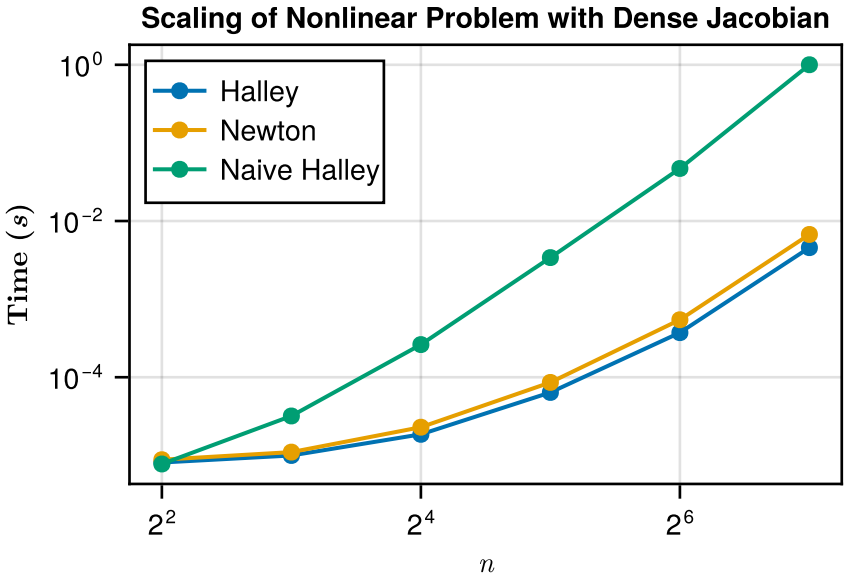

Figure 2. Scaling for solving a nonlinear problem that has a dense Jacobian, at different problem sizes. Solvers: Newton, Halley (Taylor-mode AD), and Naive Halley (full Hessian).

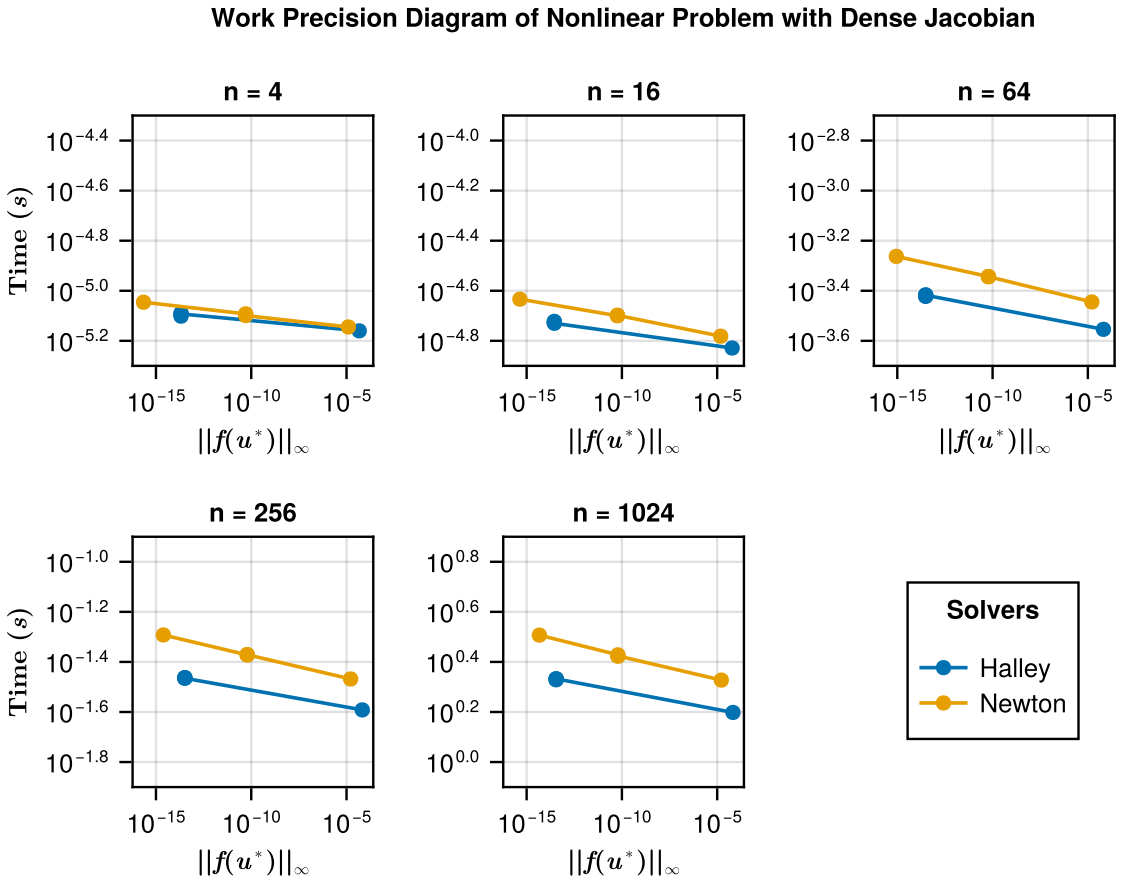

Figure 3. Work-precision diagram for solving a nonlinear problem that has a dense Jacobian, with different sizes. Solvers: Newton and Halley (Taylor-mode AD).

carried out for $K = 4, 8, 16, 32, 64, 128$ grids on x and y directions, and the Laplacian operator Δ is implemented with finite differences. Therefore, each of u and v discretizes to K^2 variables, and the total problem size of the nonlinear system is $n = 2K^2$. Since the variables only interact with nearby grids, the resulting Jacobian is sparse; in addition, when K increases, the coefficients of the finite difference operator become larger and make the Jacobian ill-conditioned.

In order to handle these characteristics, we first try to generate a sparse matrix for Jacobian in order to make factorization easier(Averick et al. 1994). This could be achieved with sparse detection techniques, either in a numerical way(Giering, Kaminski, and Slawig 2005; Walther and Griewank 2009) or in a symbolic way(Gowda, Ma, Churavy, et al. 2019). After sparsity detection, we further use the graph coloring algorithm(Gebremedhin, Manne, and Pothen 2005) to compute the matrix. These tools are available in SparseDiffTools.jl(*JuliaDiff/SparseDiffTools.jl* 2024) and SparseConnectivityTracer.jl(Hill and Dalle 2024) as a part of the Julia community. Finally, factorization of the sparse Jacobian is appropriately handled with LinearSolve.jl(*SciML/LinearSolve.jl* 2024) which has reasonable polyalgorithms for handling ill-conditioned sparse matrices. In addition, we also include the naive method of computing dense Jacobian and then doing dense factorization. The results are shown in fig. 4.

Clearly, the naive method of computing dense Jacobian and then doing dense factorization is inefficient for large problems, while the sparse method scales better. In addition, Halley's method is more efficient than Newton's method for all problem sizes and for both sparse and dense handling of the Jacobian matrix, and the advantage becomes more significant as the problem size increases. For example, for $K = 32$ and corresponding problem size $n = 2048$, Halley's method is approximately 25% faster than Newton's method.

4.3 Solving stiff ODEs

We then consider the time-dependent PDE, where we compute the time evolution of the Brusselator system given initial conditions and periodic boundary conditions above. In order to implement, the PDE is again spatially discretized with K grids on x and y directions, turning it into an ODE. Due to the Laplacian operator, the linear part of this equation has a high frequency, while the nonlinear part has a low frequency. Therefore, the resulting ODE is stiff. In solving stiff ODEs, implicit solvers are often used to ensure accuracy and efficiency(Kim et al. 2021), and these solvers requires nonlinear solve steps. Since the nonlinear solve steps are often the bottleneck of implicit solvers, impoving the nonlinear solver could accelerate the whole ODE solving process.

We choose problem size $K = 8$, $n = 2K^2 = 128$ to construct the ODE and solve with several common stiff solvers from $t_0 = 0$ to $t_e = 11.5$, and in each of the stiff solvers we test both Newton's method and Halley's method for the underlying nonlinear solver. When solving the ODE, we provide different tolerances which will determine the time step by applying adaptive time-step strategies available in DifferentialEquations.jl(Rackauckas and Nie 2017). We then use a very small tolerance to obtain very accurate solutions to the ODE, namely u_{ref} and v_{ref}, and we define the error to be the relative L^2 error of the final solution at t_e on domain $D = [0,1] \times [0,1]$, i.e.

$$\frac{\int_D \left(|\delta u(x,y,t_e)|^2 + |\delta v(x,y,t_e)|^2\right) \mathrm{d}x\mathrm{d}y}{\int_D \left(|u_{\text{ref}}(x,y,t_e)|^2 + |v_{\text{ref}}(x,y,t_e)|^2\right) \mathrm{d}x\mathrm{d}y} \tag{7}$$

where $\delta u = u - u_{\text{ref}}$ and $\delta v = v - v_{\text{ref}}$ respectively. The results are shown in fig. 5.

In the work-precision diagram, we observe that Halley's method is more efficient than Newton's method for many kinds of implicit solvers, and on average it gives a 5-10% speedup for the whole ODE solving process.

5 Conclusions

In this paper, we have demonstrated that higher-order nonlinear solvers can be implemented efficiently and scalable with higher-order automatic differentiation. We have implemented Householder's method with arbitrary convergence order and investigated the trade-off between computational cost and accelerated convergence. We have implemented $p = 2$ variant, Halley's method, for multivariate functions and have shown that Halley's method can be more efficient than Newton's method for several cases, including problems with dense Jacobian, problems with sparse Jacobian, as well as being integrated as a part of stiff ODE solvers. Backed by the Julia language and NonlinearSolve.jl framework, this method can be applied to a wide range of problems in science and engineering, and we believe that it could be a valuable tool for the scientific community.

The most noticeable limitation of this work is that we have not yet considered the case where the linear equations in each iteration must be solved iteratively, such as the case that the Jacobian is a matrix-free operator, which is common in even larger systems(Knoll and Keyes 2004). In this case, no factorization could be reused between two linear solves required by Halley's method, and the cost of each iteration could be twice as expensive as Newton's method, cancelling out the advantage of higher convergence order. In future work, we might consider constructing alternative versions of Halley's method that could reuse Krylov iterations of large linear solves.

Acknowledgements

This material is based upon work supported by the U.S. National Science Foundation under award Nos CNS-2346520, PHY-2028125, RISE-2425761, DMS-2325184, OAC-2103804, and OSI-2029670, by the Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR00112490488, by the Department of Energy,

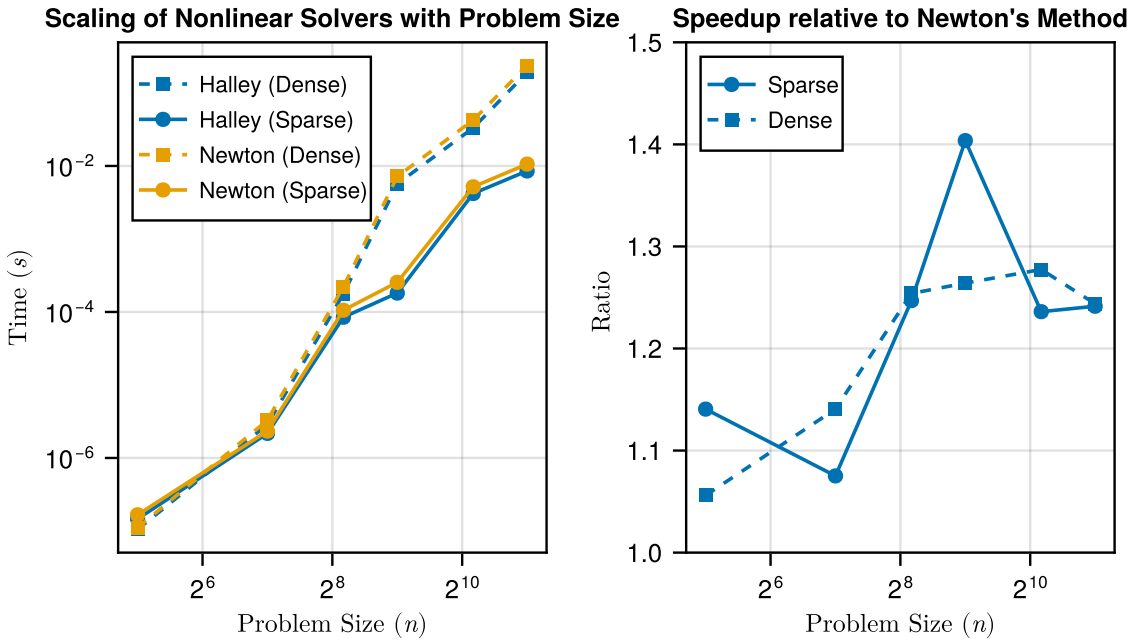

Figure 4. Scaling for solving discretized two-dimensional Brusselator steady-state problem with different problem sizes. The problems are solved to default tolerance. Left: computation time for Newton and Halley's method, each with or without sparsity detection, for each problem size. Right: relative speedup of Halley's method over Newton's method with or without sparsity detection, for each problem size.

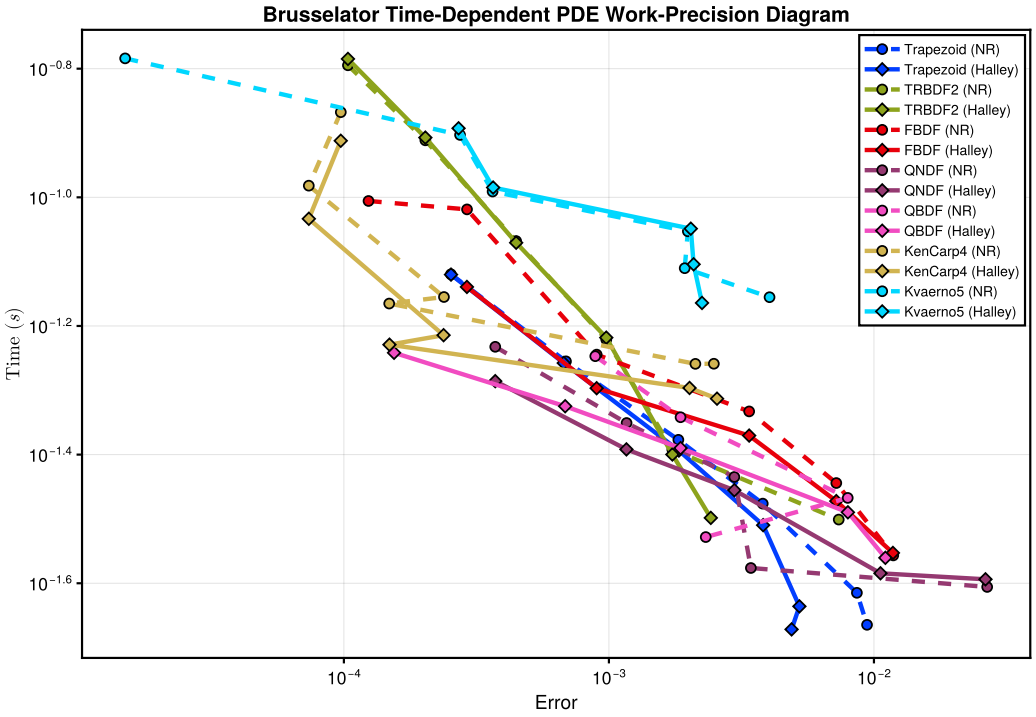

Figure 5. Work-precision diagram for solving discretized two-dimensional Brusselator time-dependent PDE. Implicit Solvers: Trapezoid(Vladimirescu 1994), TRBDF2(Hosea and L. F. Shampine 1996), FBDF(L. F. Shampine 2002), QNDF(Lawrence F. Shampine and Reichelt 1997), QBDF (alias of QNDF with $\kappa = 0$), KenCarp4(Kennedy and Carpenter 2003), Kvaerno5(Kværnø 2004). For each solver, the Newton's method is drawn in circle and solid line, and the Halley's method is drawn in diamond and dashed line.

National Nuclear Security Administration under Award Number DE-NA0003965 and by the United States Air Force Research Laboratory under Cooperative Agreement Number FA8750-19-2-1000. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government.

References

Albeanu, Grigore (2008). "On the generalized Halley method for solving nonlinear equations". In: *ROMAI J* 4.2, pp. 1–6.

Alefeld, Georg (1981). "On the convergence of Halley's Method". In: *The American Mathematical Monthly* 88.7, pp. 530–536.

Ames, William F (2014). *Numerical methods for partial differential equations*. Academic press.

Atkinson, Kendall E (1992). "A survey of numerical methods for solving nonlinear integral equations". In: *The Journal of Integral Equations and Applications*, pp. 15–46.

Averick, Brett M et al. (1994). "Computing large sparse Jacobian matrices using automatic differentiation". In: *SIAM Journal on Scientific Computing* 15.2, pp. 285–294.

Bettencourt, Jesse, Matthew J. Johnson, and David Duvenaud (2022-07). "Taylor-Mode Automatic Differentiation for Higher-Order Derivatives in JAX". en. In: URL: https://openreview.net/forum?id=SkxEF3FNPH (visited on 2022-12-22).

Bezanson, Jeff, Jake Bolewski, and Jiahao Chen (2018). "Fast flexible function dispatch in Julia". In: *arXiv preprint arXiv:1808.03370*.

Bezanson, Jeff, Alan Edelman, et al. (2017-01). "Julia: A Fresh Approach to Numerical Computing". In: *SIAM Review* 59.1. Publisher: Society for Industrial and Applied Mathematics, pp. 65–98. ISSN: 0036-1445. DOI: 10.1137/141000671. URL: https://epubs.siam.org/doi/10.1137/141000671 (visited on 2022-12-22).

Byrne, George D and Alan C Hindmarsh (1987). "Stiff ODE solvers: A review of current and coming attractions". In: *Journal of Computational physics* 70.1, pp. 1–62.

Chandrasekhar, Subrahmanyan (2013). *Radiative transfer*. Courier Corporation.

Cuyt, Annie and Paul van der Cruyssen (1983-01). "Abstract Padé-approximants for the solution of a sytem of nonlinear equations". In: *Computers & Mathematics with Applications* 9.4, pp. 617–624. ISSN: 0898-1221. DOI: 10.1016/0898-1221(83)90119-0. URL: https://www.sciencedirect.com/science/article/pii/0898122183901190 (visited on 2024-02-22).

Cuyt, Annie A. M. and L. B. Rall (1985-03). "Computational implementation of the multivariate Halley method for solving nonlinear systems of equations". en. In: *ACM Transactions on Mathematical Software* 11.1, pp. 20–36. ISSN: 0098-3500, 1557-7295. DOI: 10.1145/3147.3162. URL: https://dl.acm.org/doi/10.1145/3147.3162 (visited on 2024-02-20).

Cuyt, Annie AM (1982). "Numerical stability of the Halley-iteration for the solution of a system of nonlinear equations". In: *Mathematics of Computation* 38.157, pp. 171–179.

Galántai, Aurel (2000). "The theory of Newton's method". In: *Journal of Computational and Applied Mathematics* 124.1-2, pp. 25–44.

Gebremedhin, Assefaw Hadish, Fredrik Manne, and Alex Pothen (2005-01). "What Color Is Your Jacobian? Graph Coloring for Computing Derivatives". In: *SIAM Review* 47.4. Publisher: Society for Industrial and Applied Mathematics, pp. 629–705. ISSN: 0036-1445. DOI: 10.1137/S0036144504444711. URL: https://epubs.siam.org/doi/abs/10.1137/S0036144504444711 (visited on 2024-12-09).

Germani, A. et al. (2006-12). "Higher-Order Method for the Solution of a Nonlinear Scalar Equation". en. In: *Journal of Optimization Theory and Applications* 131.3, pp. 347–364. ISSN: 1573-2878. DOI: 10.1007/s10957-006-9154-0. URL: https://doi.org/10.1007/s10957-006-9154-0 (visited on 2024-10-09).

Giering, R., T. Kaminski, and T. Slawig (2005-10). "Generating efficient derivative code with TAF: Adjoint and tangent linear Euler flow around an airfoil". In: *Future Generation Computer Systems* 21.8, pp. 1345–1355. ISSN: 0167-739X. DOI: 10.1016/j.future.2004.11.003. URL: https://www.sciencedirect.com/science/article/pii/S0167739X04001785 (visited on 2024-12-09).

Gowda, Shashi, Yingbo Ma, Alessandro Cheli, et al. (2021-09). "High-performance symbolic-numerics via multiple dispatch". en. In: *ACM Communications in Computer Algebra* 55.3, pp. 92–96. ISSN: 1932-2240. DOI: 10.1145/3511528.3511535. URL: https://dl.acm.org/doi/10.1145/3511528.3511535 (visited on 2022-12-22).

Gowda, Shashi, Yingbo Ma, Valentin Churavy, et al. (2019-09). "Sparsity Programming: Automated Sparsity-Aware Optimizations in Differentiable Programming". en. In: URL: https://openreview.net/forum?id=rJlPdcY38B (visited on 2024-12-09).

Hill, Adrian and Guillaume Dalle (2024-11). *SparseConnectivityTracer.jl*. DOI: 10.5281/zenodo.14216520. URL: https://zenodo.org/records/14216520 (visited on 2024-12-09).

Hosea, M. E. and L. F. Shampine (1996-02). "Analysis and implementation of TR-BDF2". In: *Applied Numerical Mathematics*. Method of Lines for Time-Dependent Problems 20.1, pp. 21–37. ISSN: 0168-9274. DOI: 10.1016/0168-9274(95)00115-8. URL: https://www.sciencedirect.com/science/article/pii/0168927495001158 (visited on 2024-12-09).

Householder, Alston Scott (1970). "The numerical treatment of a single nonlinear equation". In: *(No Title)*.

JuliaDiff/SparseDiffTools.jl (2024-11). original-date: 2019-03-27T19:57:51Z. URL: https://github.com/JuliaDiff/SparseDiffTools.jl (visited on 2024-12-09).

Kelley, Carl T (2003). *Solving nonlinear equations with Newton's method*. SIAM.

Kennedy, Christopher A. and Mark H. Carpenter (2003-01). "Additive Runge–Kutta schemes for convection–diffusion–reaction equations". In: *Applied Numerical Mathematics* 44.1, pp. 139–181. ISSN: 0168-9274. DOI: 10.1016/S0168-9274(02)00138-1. URL: https://www.sciencedirect.com/science/article/pii/S0168927402001381 (visited on 2024-12-09).

Kim, Suyong et al. (2021). "Stiff neural ordinary differential equations". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 31.9. Publisher: AIP Publishing LLC, p. 093122.

Knoll, D.A. and D.E. Keyes (2004). "Jacobian-free Newton–Krylov methods: a survey of approaches and applications". In: *Journal of Computational Physics* 193.2, pp. 357–397. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2003.08.010. URL: https://www.sciencedirect.com/science/article/pii/S0021999103004340.

Kunkel, Peter (2006). *Differential-algebraic equations: analysis and numerical solution*. Vol. 2. European Mathematical Society.

Kværnø, A. (2004-08). "Singly Diagonally Implicit Runge–Kutta Methods with an Explicit First Stage". en. In: *BIT Numerical Mathematics* 44.3, pp. 489–502. ISSN: 1572-9125. DOI: 10.1023/B:BITN.0000046811.70614.38. URL: https://doi.org/10.1023/B:BITN.0000046811.70614.38 (visited on 2024-12-09).

Madhu, Kalyanasundaram and Jayakumar Jayaraman (2017). "Some higher order Newton-like methods for solving system of nonlinear equations and its applications". In: *International Journal of Applied and Computational Mathematics* 3.3, pp. 2213–2230.

Margossian, Charles C (2019). "A review of automatic differentiation and its efficient implementation". In: *Wiley interdisciplinary reviews: data mining and knowledge discovery* 9.4, e1305.

Noor, Khalida Inayat, Muhammad Aslam Noor, and Shaher Momani (2007). "Modified Householder iterative method for nonlinear equations". In: *Applied mathematics and computation* 190.2, pp. 1534–1539.

Ogbereyivwe, Oghovese, Orobosa Izevbizua, and Salisu Shehu Umar (2023). "Some high-order convergence modifications of the Householder method for nonlinear equations". In: *Communications in Nonlinear Analysis* 11.2, pp. 1–11.

Pal, Avik et al. (2024-03). *NonlinearSolve.jl: High-Performance and Robust Solvers for Systems of Nonlinear Equations in Julia*. en. arXiv:2403.16341 [cs, math]. URL: http://arxiv.org/abs/2403.16341 (visited on 2024-06-15).

Potra, FA (1984). "Nondiscrete induction and iterative processes". In: *Pitman Publ.*

Prigogine, Ilya and René Lefever (1968). "Symmetry breaking instabilities in dissipative systems. II". In: *The Journal of Chemical Physics* 48.4, pp. 1695–1700.

Rackauckas, Christopher and Qing Nie (2017). "Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia". In: *Journal of Open Research Software* 5.1, p. 15.

Revels, Jarrett, Miles Lubin, and Theodore Papamarkou (2016). "Forward-mode automatic differentiation in Julia". In: *arXiv preprint arXiv:1607.07892*.

Sariman, Syahmi Afandi and Ishak Hashim (2020). "New optimal Newton-Householder methods for solving nonlinear equations and their dynamics". In: *Computers, Materials & Continua* 65.1, pp. 69–85.

SciML/LinearSolve.jl (2024-12). original-date: 2021-07-02T19:56:38Z. URL: https://github.com/SciML/LinearSolve.jl (visited on 2024-12-09).

Shampine, L. F. (2002-12). "Solving 0 = F(t, y(t), y'(t)) in Matlab". en. In: 10.4. Publisher: De Gruyter Section: Journal of Numerical Mathematics, pp. 291–310. ISSN: 1569-3953. DOI: 10.1515/JNMA.2002.291. URL: https://www.degruyter.com/document/doi/10.1515/JNMA.2002.291/html (visited on 2024-12-09).

Shampine, Lawrence F. and Mark W. Reichelt (1997-01). "The MATLAB ODE Suite". en. In: *SIAM Journal on Scientific Computing* 18.1, pp. 1–22. ISSN: 1064-8275, 1095-7197. DOI: 10.1137/S1064827594276424. URL: http://epubs.siam.org/doi/10.1137/S1064827594276424 (visited on 2024-12-09).

Singh, Harmandeep and Janak Raj Sharma (2023-02). "Simple yet highly efficient numerical techniques for systems of nonlinear equations". en. In: *Computational and Applied Mathematics* 42.1, p. 22. ISSN: 2238-3603, 1807-0302. DOI: 10.1007/s40314-022-02159-9. URL: https://link.springer.com/10.1007/s40314-022-02159-9 (visited on 2024-06-03).

Steihaug, Trond (2006). "Newton and Halley are one step apart". In: *Institute for Scientific Computing*.

Suleiman, Sara Tagelsir Mohamed (2009). "Solving System of Nonlinear Equations Using Methods in the Halley Class". MA thesis. The University of Bergen.

Tan, Songchen (2023a-06). "Higher-Order Automatic Differentiation and Its Applications". en. MA thesis. Massachusetts Institute of Technology. URL: https://dspace.mit.edu/handle/1721.1/151501.

Tan, Songchen (2023b-05). *TaylorDiff.jl*. original-date: 2022-11-09T17:21:07Z. URL: https://github.com/JuliaDiff/TaylorDiff.jl (visited on 2023-05-11).

Vladimirescu, Andre (1994-03). *The Spice Book*. USA: John Wiley & Sons, Inc. ISBN: 978-0-471-60926-1.

Walther, Andrea and Andreas Griewank (2009). "Getting Started with ADOL-C." In: *Combinatorial scientific computing* 1.06.02.

White, Frames et al. (2023-04). *JuliaDiff/ChainRules.jl: v1.49.0*. DOI: 10.5281/zenodo.7870094. URL: https://zenodo.org/record/7870094 (visited on 2023-05-11).

Xiao, Xiao-Yong (2022-09). "New techniques to develop higher order iterative methods for systems of nonlinear equations". en. In: *Computational and Applied Mathematics* 41.6, p. 243. ISSN: 2238-3603, 1807-0302. DOI: 10.1007/s40314-022-01959-3. URL: https://link.springer.com/10.1007/s40314-022-01959-3 (visited on 2024-05-17).

Xiao, Xiaoyong and Hongwei Yin (2015-08). "A new class of methods with higher order of convergence for solving systems of nonlinear equations". In: *Applied Mathematics and Computation* 264, pp. 300–309. ISSN: 0096-3003. DOI: 10.1016/j.amc.2015.04.094. URL: https://www.sciencedirect.com/science/article/pii/S0096300315005561 (visited on 2024-10-09).